Capítulo
7

Estimação de Esforço

Este capítulo apresenta algumas técnicas importantes e largamente usadas para estimar o esforço de desenvolvimento de software, porque não se pode planejar um projeto sem saber quanto tempo ele vai durar e quanto vai custar. Inicialmente (Seção 7.1), é apresentada a técnica paramétrica mais usada e padronizada no mundo, *Análise de Pontos de Função*. Na sequência, a Seção 7.2 apresenta a técnica de *Pontos de História*, bastante usada por equipes ágeis. Em seguida, na Seção 7.3, são apresentados os conceitos de contagens de linhas de código, KSLOC, o que leva à técnica COCOMO II ou CII, apresentada na Seção 7.4. O Capítulo 14, que trata das atividades de manutenção e evolução de software aborda também técnicas de estimação de esforço específicas para as atividades de manutenção.

Uma das questões fundamentais em um projeto de software é saber, antes de executá-lo, quanto esforço, em horas, dias ou meses de trabalho, será necessário para levá-lo a termo. Essa área, chamada de *estimação de esforço*, conta com algumas técnicas que, apesar das críticas, têm apresentado resultados interessantes ao longo dos últimos anos.

A maioria das técnicas de estimação de esforço utiliza pelo menos um parâmetro como base, por isso elas são chamadas de *técnicas paramétricas*.

Um exemplo de técnica *não paramétrica* é a *estimação por especialista*. Neste caso, simplesmente pede-se a pessoas com bastante experiência em desenvolvimento de software que deem sua opinião sobre o tempo e esforço total de desenvolvimento. Dificilmente estimações precisas são obtidas por essa ou mesmo por outras técnicas, mas usualmente, quando se tem pelo menos uma ordem de magnitude para o tempo e esforço de desenvolvimento pode-se ir negociando escopo e cronograma ao longo do projeto com menos desgaste.

Existem também técnicas *paramétricas* baseadas em alguma medida objetiva que possa ser feita sobre o sistema que foi ou vai ser desenvolvido, como, por exemplo, o número de linhas do programa, seus requisitos, descritos como funções, casos de uso ou histórias de usuário.

As técnicas de pontos de função, pontos de caso de uso e pontos de história baseiam-se em um conjunto de requisitos, aos quais é atribuído um peso que determinará, a partir de certas transformações matemáticas, o esforço necessário para seu desenvolvimento.

Já as técnicas baseadas em linhas de código, como CII, necessitam de uma estimativa de quantas linhas de código deverão ser produzidas. Existem também mecanismos que permitem converter parâmetros como pontos de função em estimativas de linhas de código e vice-versa, fazendo que as técnicas acabem sendo compatíveis entre si.

Neste capítulo, inicialmente, é descrita a técnica de pontos de função, devido ao seu largo uso. Na sequência são apresentadas as técnicas de pontos de história e CII. A descrição da técnica de *pontos de caso de uso*, que aparecia na primeira edição deste livro agora é descrita com mais detalhes no Capítulo 4 de Wazlawick (2015). Esta técnica, apesar de ser simples de aplicar e de ser compatível com UML e o Processo Unificado, ainda não dispõe de tanta comprovação empírica quanto CII e Análise de Pontos de Função.

7.1 Análise de Pontos de Função

A técnica de *Análise de Pontos de Função* (APF), ou *Function Point Analysis* (Albrecht & Gaffney Jr., 1983), é uma técnica paramétrica de estimativa de esforço para desenvolvimento de software que usa como parâmetro os conceitos de funções de dados e funções transacionais, que basicamente correspondem aos requisitos funcionais de um sistema.

A análise de pontos de função é aplicável, portanto, a partir do momento em que os requisitos funcionais do software tenham sido definidos. Esses requisitos serão convertidos em valores numéricos, que, depois de ajustados à capacidade da empresa desenvolvedora, representarão o esforço necessário para desenvolver o sistema. Assim, a medida obtida pela técnica é independente da linguagem de programação e da tecnologia empregada.

A APF pode ser aplicada para medir o tamanho de um sistema antes de desenvolvê-lo, de forma que seu custo seja previsto mais adequadamente. Além disso, pode ser aplicada também a processos de manutenção, permitindo estimar o esforço necessário para implantar determinada alteração no software, especialmente se for a adição de uma nova funcionalidade.

A técnica também pode ser usada para calcular o custo-benefício de software ou componentes de software comprados, uma vez que a divisão do preço do produto pelo número de pontos de função que ele implementa dá uma ideia de seu custo relativo. Então, o custo por ponto de função pode ser visto como um fator de normalização para a comparação de produtos de software.

Como é baseada em requisitos implementados, e não no número de linhas de código produzidas, a técnica é mais adequada para medir a produtividade de uma equipe de desenvolvimento, pois ferramentas de produtividade e reusabilidade permitirão implementar mais funcionalidades perceptíveis para o cliente com menos linhas de código sendo escritas.

Existem três níveis de contagem, dependendo da quantidade de informações que se tenha sobre o sistema que são a *indicativa*, a *estimada* e a *detalhada*. Inicialmente apresentaremos a detalhada, que explora todos os conceitos relacionados a pontos de função e depois as duas outras formas de contagem que são simplificações desta.

Em relação ao *propósito* da contagem, existem três situações específicas que são:

- *Contagem para desenvolvimento de projeto*: é usada para estimar o esforço para o desenvolvimento de um novo projeto. Esta é a situação predominante nas descrições deste capítulo.
- *Contagem para melhoria de projeto*: é usada para a evolução de software, em que se contam as funcionalidades adicionadas, alteradas e removidas. A técnica é normalmente aplicável apenas para a manutenção adaptativa, já que a manutenção corretiva e a manutenção perfectiva são muito imprevisíveis (Seção 14.2).
- *Contagem de aplicação*: é usada para contar pontos de função de aplicações existentes. Pode ter vários objetivos, dentre eles estimar o tamanho funcional da aplicação de forma a relativizar outras métricas (Seção 9.5). Pode, por exemplo, ser mais realista conhecer o número de defeitos por ponto de função do que simplesmente o número de defeitos total do software.

A contagem de pontos de função segue um método composto pelos seguintes passos, que são explicados detalhadamente depois:

- Determinar o tipo de contagem (desenvolvimento, melhoria ou aplicação existente).
- Determinar os limites ou fronteiras da aplicação e o seu escopo.
- Identificar e atribuir valor em pontos de função para as funções de dados (arquivos lógicos internos e arquivos de interface externa).
- Identificar e atribuir valor em pontos de função para funções transacionais (entradas externas, consultas externas e saídas externas).

A técnica é divulgada e normatizada internacionalmente pelo IFPUG (*International Function*

Point Users Group) e no Brasil pelo BFPUG (*Brazilian Function Point Users Group*). APF é reconhecida como métrica de software pela ISO na norma ISO/IEC 20926 – *Software Engineering – Function Point Counting Practices Manual*.

Além do método de contagem do IFPUG, existem outros dois métodos internacionalmente relevantes: NESMA, da associação holandesa de métricas, e Mark II (Symons, 1988), ou MK II, mantido pela associação inglesa de métricas. Ao contrário do manual de contagem do IFPUG, que deve ser adquirido, os manuais dessas duas técnicas podem ser obtidos gratuitamente em seus *sites*, bastando, para isso, fazer registro gratuito na respectiva associação.

Neste capítulo, a apresentação baseia-se nas orientações do Ministério do Planejamento do Governo Brasileiro (Bomfim & Andrade, 2015), que complementa as informações do manual de contagem do IFPUG com orientações adicionais e adota terminologia em língua portuguesa e do roteiro de métricas do SERPRO (Hazan *et al.*, 2010), visto que estes dois documentos são largamente utilizados como referência no território nacional para desenvolvimento de projetos na área pública e também privada. Mais detalhes e exemplos de aplicação da técnica podem ser encontrados nestes documentos acessíveis a partir dos *QR codes* ao lado [QRC 7.1] [QRC 7.2]

7.1.1 Interpretação e Classificação de Requisitos como Funções

A APF é baseada na contagem de pontos para cada uma das funções do sistema. Primeiramente deve-se ter em mente que apenas funcionalidades visíveis para o usuário devem ser contadas. Então, não é todo e qualquer requisito que conta. Se um requisito menciona apenas algum cálculo interno, não deve ser contado como função, embora possivelmente vá aparecer como parte de alguma outra função visível para o usuário.

Apenas transferências de informação para dentro e para fora da fronteira do sistema (e arquivos de dados lógicos reconhecíveis pelo usuário) são considerados funções.

Um requisito pode ter mais de uma função representada nele. Se o requisito trata, por exemplo, de manter um cadastro de clientes e de produtos, então, na verdade, são dois cadastros, que devem ser interpretados como dois arquivos individuais, cada qual com sua contagem de pontos. E mais ainda, cada cadastro vai comportar um certo número de funções transacionais para consultar e alterar seus registros, as quais são computadas individualmente.

Assim, um primeiro passo para o uso da técnica é tomar os requisitos, eliminar aqueles que são meramente funções internas (ou incorporá-los às funções a que eventualmente pertençam) e subdividir aqueles que representam mais de uma função, até obter uma lista de funções individuais visíveis para o usuário.

Embora trabalhos como o de Longstreet (2012) demonstrem como interpretar essas funções em termos de elementos de interface com o usuário, elas também podem ser identificadas nos requisitos funcionais, ou seja, na sua forma mais essencial e independente de tecnologia.

A técnica APF avalia as duas naturezas de funções:

- *Funções de dados*, ou seja, a representação estrutural dos dados, na forma de arquivos lógicos internos (ALI) e arquivos de interface externas (AIE).
- *Funções transacionais*, ou seja, a representação das transações sobre os dados, na forma de entradas externas (EE), consultas externas (CE) e saídas externas (SE).

As funções transacionais (EE, CE e SE) devem ser processos elementares, isso é, de único passo. Uma transação é a menor unidade de atividade que faça sentido do ponto de vista do usuário e, o

que é mais importante, deixe o sistema em um estado consistente.

De outro lado, os ALI e AIE devem ser elementos complexos de informação reconhecíveis pelo usuário (ou seja, não são necessariamente representações físicas internas). Podem ser considerados arquivos internos e externos os elementos de informação que possam ser representados por classes do modelo conceitual ou tabelas do modelo relacional, por exemplo. Porém, apenas os principais arquivos que representam entidades reconhecíveis devem ser contados. Não se deve contar classes que não representem entidades, como classes de controle ou de interface. Também não se deve contar tabelas auxiliares em bancos de dados como tabelas que representam relações $n \times n$ ou dados temporários.

As subseções seguintes discutem em mais detalhes os cinco tipos de transações.

7.1.1.1 ALI - Arquivo Lógico Interno

Um *arquivo lógico interno* (ALI) pode ser considerado como um elemento do modelo conceitual percebido pelo usuário e mantido internamente pelo sistema. Ele não pode ser parte de qualquer agregação ou composição. O arquivo lógico interno é uma informação complexa (uma classe, tabela, conceito ou entidade) do tipo AL, ou seja, uma classe que não é componente de outras classes, embora possa ter seus próprios componentes.

Arquivos lógicos internos (ALI) são aqueles mantidos pela própria aplicação, enquanto arquivos de interface externa (AIE) são aqueles usados pela aplicação, mas mantidos externamente por outras aplicações, por exemplo, dados obtidos através de um *Web service* externo à aplicação.

O que vai determinar se um arquivo é interno ou externo, portanto, é a definição da *fronteira* ou *borda do sistema*, ou seja, os limites daquilo que é considerado interno ao sistema que está sendo desenvolvido ou analisado e de outros sistemas. No caso de aplicações cliente/servidor, deve-se considerar que a borda do sistema engloba ambos os módulos, porque nenhum deles, isoladamente, se constitui em uma aplicação completa com significado para o usuário.

Aqui convém fazer uma ressalva, já que o método APF original diferencia os *registros lógicos* (RL), que correspondem a um subconjunto de dados reconhecível pelo usuário dentro de um arquivo lógico (AL), seja ele interno ou externo. Fazendo-se um paralelo com orientação a objetos, os RL seriam classes quaisquer, enquanto os AL seriam classes que *não* são componentes nem agregados de outras classes. Por exemplo, se um automóvel é composto por motor, carroceria e pneus, então apenas o conceito *Automóvel* seria considerado um AL, mas *Motor*, *Carroceria* e *Pneu* seriam RLs contidas neste AL.

Assim, um AL é uma classe que, caso participe de uma composição ou agregação, ocupa o lugar mais alto da hierarquia. Um AL pode *ter* componentes, mas não pode *ser* componente. Essa distinção é importante, porque o cálculo da complexidade das funções transacionais (EE, SE e CE) é baseado em AL, enquanto o cálculo da complexidade das funções de dados (ALI e AIE) é baseado em RL. Resumindo, o cálculo de complexidade das funções transacionais contabiliza apenas as classes no mais alto nível de uma hierarquia de composição (AL), enquanto o cálculo de complexidade de funções de dados contabiliza quaisquer classes, inclusive as componentes (RL).

Arquivos lógicos internos e arquivos de interface externa, por definição devem ser do tipo AL. RLs não são considerados como funções de dados para efeito da contagem de pontos de função; são apenas considerados como parâmetros para a definição da complexidade dos ALI ou AIE aos quais pertencem.

7.1.1.2 AIE - Arquivos de Interface Externa

Um *arquivo de interface externa* (AIE) pode ser considerado como um elemento do modelo

conceitual percebido pelo usuário e mantido externamente por outras aplicações.

Um arquivo de interface externa é uma informação complexa (uma classe, tabela, conceito ou entidade) do tipo AL que é mantida em outras aplicações, ou seja, não é gerenciada pela aplicação para a qual se está contando pontos de função. Por exemplo, um sistema de caixa automático, que consulta informações de uma agência bancária pode considerar que os dados do cliente e das suas contas bancárias são arquivos de interface externa, pois não são mantidos pela aplicação do caixa automático, sendo consultadas e alteradas através de uma interface externa.

Usualmente os ALI e AIE têm associadas funções transacionais CE, SE e EE que são contabilizadas à parte da função de dados em si.

7.1.1.3 EE - Entrada Externa

Uma *entrada externa* (EE) consiste em uma entrada de dados ou controle que tem como consequência a alteração do estado interno dos dados do sistema. Entradas externas podem ser sucedidas por mensagens de confirmação ou de erro, desde que elas não se configurem em consultas aos dados.

Entradas externas são, portanto, funções que pegam dados ou controle do usuário ou de outras aplicações e levam para dentro do sistema. A função deve ter como objetivo *incluir*, *alterar* ou *excluir* dados de forma direta (alterando os dados no sistema) ou indireta (solicitando a outra aplicação que faça a alteração nos dados). Por outro lado, dados passados pelo usuário com o único fim de servir de *parâmetro* para uma consulta ou saída, mas que não provoquem alteração no estado interno do sistema, não devem ser considerados como entradas externas.

Um comando que passa o identificador de um registro (por exemplo, o CPF de um cliente) para excluir um conjunto de dados (por exemplo, excluir o cliente) pode ser considerado como entrada externa, pois é um controle que remove dados, ou seja, faz uma mudança no estado interno das informações do sistema.

Assim, entradas externas são funções transacionais que incluem, alteram ou excluem informação no sistema. Seus argumentos são os dados passados como parâmetro para localizar os objetos a serem alterados ou excluídos e os parâmetros que correspondem aos novos valores para os objetos a serem criados e alterados, quando for o caso.

Algumas vezes, as operações de alteração e exclusão ocorrem em dois passos, ou seja, são precedidas de uma consulta. Essa consulta é contabilizada à parte, como uma CE. Por exemplo, um usuário que deseja fazer a alteração do cadastro de um cliente primeiramente entra com o CPF desse cliente e visualiza seus dados na tela (CE); em seguida, ele altera os dados que deseja e salva as novas informações (EE).

Além disso, se uma entrada externa puder produzir um conjunto de mensagens de erro, por conta de possíveis exceções, cada mensagem vai contar como um argumento da entrada (conforme será visto mais adiante). Essas mensagens não devem ser contabilizadas como saídas independentes.

7.1.1.4 SE - Saída Externa

Uma *saída externa* (SE) consiste em uma saída de dados que pode ser precedida ou não da entrada de parâmetros. Pelo menos um dos dados de saída deve ser derivado, ou seja, calculado.

Saídas externas são, então, funções que pegam dados de dentro do sistema e apresentam ao usuário ou enviam a outras aplicações em sua forma original ou transformada, sendo que pelo menos um valor derivado (calculado) deve existir para que seja uma saída externa.

As SE, portanto, são os fluxos de informação para fora do sistema. Não se conta como SE uma

função que simplesmente pega dados do sistema e os apresentam da forma como estão (isso será contabilizado como uma *consulta* externa). Mas se houver qualquer tipo de operação matemática ou lógica sobre esses dados, então a função poderá ser considerada uma saída.

Uma saída externa também pode ter parâmetros. Por exemplo, passam-se o CPF de um cliente e uma data e, a partir desses parâmetros, se obtém um relatório de todas as vendas realizadas para esse cliente a partir da data definida e o seu total. Como a data e o CPF são usados para filtrar as vendas que devem aparecer no relatório e fazer a totalização, essa função deve ser considerada saída externa, e não meramente consulta externa.

7.1.1.5 CE - Consulta Externa

Uma *consulta externa* (CE) corresponde a uma saída de dados que pode ser precedida ou não da entrada de parâmetros. Os dados devem sair da mesma forma como estavam armazenados, sem transformações ou cálculos.

A *consulta externa* consiste da apresentação de dados da mesma forma como foram armazenados, sem cálculos ou transformações. Normalmente, a consulta inclui parâmetros de entrada para localizar os objetos da consulta.

Note que, quando da verificação da complexidade de todas funções transacionais ou de dados, apenas a complexidade visível pelo usuário será considerada. A complexidade algorítmica interna não é aplicada às funções individuais, até porque, a princípio, ela pode ainda não ser conhecida nesse momento da análise dos requisitos.

7.1.2 UFP – Pontos de Função Não Ajustados

Pontos de função não ajustados são a medida inicial de complexidade de um sistema. Seu valor corresponde à soma dos pontos de função atribuídos a cada uma das funções de dados ou transacionais. Se forem aplicados a eles os fatores de ajuste técnico eles se tornam pontos de função ajustados, os quais são vistos na Seção 7.1.4.

Assim, uma vez determinado o tipo da função (transacional ou de dados), sua complexidade vai ser calculada a partir dos seguintes fatores:

- *Registro Lógico* (*RL*): corresponde a um subconjunto de dados reconhecível pelo usuário dentro de um ALI ou AIE (uma classe qualquer que represente uma entidade).
- *Arquivo Lógico* (*AL*): corresponde a um ALI ou AIE, usado em uma transação (uma classe que represente uma entidade e que não seja componente de outra).
- *Tipo de dados elementar* (*TDE*): corresponde a uma unidade de informação (um campo), a princípio indivisível e reconhecível pelo usuário; normalmente seria identificado como um campo de uma tabela, um atributo de uma classe ou um parâmetro de uma função.

A complexidade das funções transacionais EE, SE e CE é determinada pela quantidade de arquivos AL e dados TDE envolvidos. Já a complexidade das funções de dados ALI e AIE é determinada pela quantidade de registros RL e argumentos TDE.

No caso das entradas externas, os TDE podem ser os campos de entrada de informação, mensagens de erro e botões que podem ser pressionados, por exemplo.

No caso das saídas externas, os TDE podem ser os campos em um relatório, valores calculados, mensagens de erro e cabeçalhos de colunas que são lidos de um arquivo interno.

No caso das consultas externas, os TDE podem ser os campos usados para pesquisa e os campos mostrados como resposta à consulta. Se houver vários botões para fazer a consulta de maneira diferente, cada um deles também deve contar como um TDE.

Note que as saídas e consultas podem tanto ter dados de entrada (parâmetros) como dados de

saída (resultados). Normalmente, as entradas externas só têm campos de entrada (a informação que se vai armazenar), mas as mensagens de erro potencialmente produzíveis por uma entrada também podem ser contabilizadas como um dado TDE.

A complexidade de uma função transacional é, portanto, calculada a partir da quantidade de AL e TDE envolvidos. O valor de *#AL* corresponde ao número de classes do modelo conceitual (que não são componentes ou agregados de outra classe) que contêm as informações de entrada e/ou saída como atributos. Por exemplo, se uma função de entrada passa uma data que será armazenada como atributo da classe *X* e um valor numérico que será armazenado como atributo de uma classe *Y*, então existem duas classes AL envolvidas e o valor de *#AL*, nesse caso, para efeito de cálculo da complexidade da função, é igual a 2.

Se os dois valores vão ser armazenados como atributos de uma mesma classe, ou como atributos de dois componentes de uma mesma classe, ou ainda como atributos de uma classe e uma de suas componentes, então considera-se que há apenas um AL. Por exemplo, uma função que permite informar o número do chassis e o modelo dos pneus de um carro tem apenas um AL, já que *Carroceria* e *Pneu* são classes agregadas à classe *Automóvel*.

O segundo valor usado para calcular a complexidade de uma função transacional é o seu número de argumentos, *#TDE*, ou seja, a quantidade de valores individuais de entrada e/ou saída, independentemente das classes a que eles pertençam. Assim, por exemplo, uma função de entrada EE, que envia cinco valores diferentes ao sistema, terá cinco argumentos, e o valor de #TDE será igual a 5. Uma consulta CE que passa dois argumentos e recebe oito valores como retorno tem #TDE igual a 10.

Deve-se ainda saber que o que conta, no caso de TDE, é o *tipo* de valor. Caso a função envie ou receba uma lista de valores de um mesmo tipo, a lista conta uma única vez. Por exemplo, uma função que passar um nome de pessoa e uma lista de telefones terá dois argumentos (mesmo que essa lista tenha dez telefones, ela conta uma única vez). De outro lado, se algum tipo de cálculo for feito com a lista de valores, então cada resultado de cálculo contará como um argumento diferente. Por exemplo, uma saída que apresente uma lista de notas, a média e o desvio-padrão dessas notas deverá ter considerados três TDE.

A Tabela 7.1 mostra como determinar a complexidade das entradas externas EE a partir dos AL e TDE. Já as saídas e consultas externas, SE e CE, têm sua complexidade determinada de acordo com a Tabela 7.2. Finalmente, os arquivos internos e externos, ALI e AIE, têm sua complexidade determinada em função de classes quaisquer (inclusive componentes de outras classes), ou seja, RL. Além do valor de RL, deve-se usar o valor de TDE, como no caso de funções transacionais para determinar a complexidade das funções de dados. Deve ser, então, aplicada a Tabela 7.3.

TABELA 7.1 Complexidade funcional de entradas externas (EE)

	#TDE		
#AL	**1 a 4**	**5 a 15**	**16 ou mais**
0 a 1	Baixa	Baixa	Média
2	Baixa	Média	Alta
3 ou mais	Média	Alta	Alta

TABELA 7.2 Complexidade funcional de saídas externas (SE) e consultas externas (CE)

	#TDE		
#AL	**1 a 5**	**6 a 19**	**20 ou mais**
0 a 1	Baixa	Baixa	Média
2 a 3	Baixa	Média	Alta
4 ou mais	Média	Alta	Alta

TABELA 7.3 Complexidade funcional de arquivos lógicos internos (ALI) e arquivos de interface externa (AIE)

	#TDE		
#RL	**1 a 19**	**20 a 50**	**51 ou mais**
1	Baixa	Baixa	Média
2 a 5	Baixa	Média	Alta
6 ou mais	Média	Alta	Alta

Observe que o número mínimo de RL envolvido com um AL deve ser 1 porque uma classe que não tem nenhum componente pode ser considerada tanto como AL quanto como RL e assim, essa classe tem necessariamente um RL envolvido que é ela própria.

Depois de determinar a complexidade de cada função transacional e função de dados, pode-se obter seu número de pontos de função não ajustados *UFP* aplicando a Tabela 7.4.

TABELA 7.4 Pontos de função não ajustados por tipo e complexidade de função

	Complexidade funcional		
Tipo de função	**Baixa**	**Média**	**Alta**
EE	3	4	6
SE	4	5	7
CE	3	4	6
ALI	7	10	15
AIE	5	7	10

O número *UFP* de pontos de função não ajustados para o sistema como um todo será simplesmente a soma dos pontos de função não ajustados obtidos para cada uma das funções do sistema.

A identificação das funções transacionais e de dados de um sistema é o ponto-chave para determinar sua complexidade. Porém, muitas vezes, requisitos são incompletos ou muito genéricos para que uma boa contagem possa ser feita. Por isso, as técnicas de contagem indicativa e estimada podem ser usadas, conforme explicado adiante.

Porém, a própria técnica de pontos de função pode ajudar a melhorar a qualidade dos requisitos.

Em primeiro lugar, ela precisa dos valores TDE; então será necessário verificar se os requisitos efetivamente listam todas as informações elementares necessárias para realizar cada função.

Em segundo lugar, a identificação de funções de dados e transacionais caminham juntas. Uma vez identificado um ALI ou AIE (classe), pode-se imaginar que pelo menos as transações mais básicas para esse arquivo devam existir. Essas transações, usualmente chamadas CRUD ou CRUDL (acrônimo para as cinco operações: *create*, *retrieve*, *update*, *delete* e *list*), implicam em inclusão (EE), alteração (EE), remoção (EE), consulta (CE) e listagem (CE), nem sempre essas funções estão todas presentes na aplicação para cada arquivo lógico. É possível, por exemplo, que a aplicação crie um arquivo de dados no qual só se possa inserir e consultar, mas não alterar ou excluir dados. Deve-se sempre analisar quais funções transacionais realmente devem existir ou não para cada arquivo.

Além disso, nem sempre um arquivo sofre apenas transações tão básicas em seu ciclo de vida. Em uma livraria virtual, por exemplo, um livro não será apenas incluído, alterado, consultado e removido; em determinados momentos ele será colocado em oferta, reservado ou vendido. Essas transações podem ser consideradas simples alterações, já incluídas no CRUDL?

Se uma transação de alteração de um objeto, como "colocar em oferta", implica usar apenas um conjunto ou um subconjunto de TDE e os arquivos AL ou um subconjunto deles, já considerados na operação de alteração do CRUDL, então a alteração já é contemplada pelas transações CRUDL. Por exemplo, se colocar um livro em oferta implica simplesmente mudar o valor de um de seus atributos, o que já pode ser feito pela operação CRUDL de alteração, então "colocar um livro em oferta" não será uma nova função nem vai contar pontos de função, mas será considerado um caso especial da alteração de livro no CRUDL.

De outro lado, se a alteração implica usar dados TDE não incluídos no conjunto da operação de alteração ou ainda arquivos AL não incluídos na alteração do CRUDL, então a transação deverá ser considerada uma nova função e contar seus próprios pontos de função. Por exemplo, se "colocar livro em oferta" implica obter dados de um arquivo de ofertas padrão, o que não é feito quando da operação de alteração do CRUDL, então trata-se de uma nova função.

A título de exemplo de aplicação da técnica, considere a seguinte lista de requisitos:

- R1: O sistema deve permitir o gerenciamento (CRUDL) de informações sobre livros e usuários. Dos livros incluem-se título, ISBN, autor, número de páginas, editora e ano de publicação. Dos usuários incluem-se nome, documento, endereço, telefone e *e-mail*.
- R2: O sistema deve permitir o registro de empréstimos, em que são informados o documento do usuário e o ISBN de cada livro.
- R3: Quando um empréstimo for executado, o sistema deve armazenar as informações em uma tabela relacional, usando chaves estrangeiras para identificar o usuário e os livros.
- R4: Após o registro de um empréstimo, deve ser impresso um recibo com o nome do usuário, além de título e data de devolução prevista para cada livro, que deve ser calculada como a data atual somada ao prazo do livro.

Analisando-se o primeiro requisito, percebe-se que se trata de dois cadastros independentes com suas operações CRUDL. O segundo requisito é uma entrada externa. O terceiro requisito é uma função interna que não deve ser contabilizada. O quarto requisito será considerado uma saída externa, pois realiza um cálculo para obter a data de entrega em função da data atual e do prazo do livro. As funções a serem consideradas são apresentadas na Tabela 7.5.

TABELA 7.5 Exemplo de identificação de funções de dados e transacionais a partir de requisitos

Função	Tipo	AL	RL	TDE	#AL	#RL	#TDE	Complex.	UFP

Classe Livro	ALI		Livro	Título, ISBN, autor, número de páginas, editora, ano de publicação, preço		1	7	Baixa	7
-Inserir livro	EE	Livro, editora, autor		Título, ISBN, autor, número de páginas, editora, ano de publicação, preço	3		7	Alta	6
-Alterar livro	EE	Livro		Preço	3		1	Baixa	3
-Excluir livro	EE	Livro		Título, ISBN	1		2	Baixa	3
-Consultar livro	CE	Livro, editora, autor		Título, ISBN, autor, número de páginas, editora, ano de publicação, preço	3		7	Média	4
-Listar livros	CE	Livro, editora, autor		Título, ISBN, autor, número de páginas, editora, ano de publicação, preço	3		7	Média	4
Classe Pessoa	ALI		Pessoa	Nome, documento, endereço, telefone, *e-mail*		1	5	Baixa	7
-Inserir usuário	EE	Pessoa		Nome, documento, endereço, telefone, *e-mail*	1		5	Baixa	3
-Alterar usuário	EE	Pessoa		Nome, endereço, telefone, *e-mail*	1		4	Baixa	3
-Excluir usuário	EE	Pessoa		Nome, documento	1		2	Baixa	3
-Consultar usuário	CE	Pessoa		Nome, documento, endereço, telefone, *e-mail*	1		5	Baixa	3
-Listar usuários	CE	Pessoa		Nome, documento	1		2	Baixa	3
Registrar empréstimo	EE	Pessoa, livro, empréstimo		Nome da pessoa, ISBN dos livros	3		2	Média	4
Imprimir recibo	SE	Pessoa, livro, empréstimo		Nome da pessoa, título do livro, data de hoje, prazo do livro, data de devolução	3		5	Baixa	4
TOTAL									**57**

Portanto, aplicando-se a técnica às funções identificadas nos requisitos mencionados, considerando que o modelo conceitual tenha as classes *Pessoa*, *Livro*, *Empréstimo*, *Editora* e *Autor*, sem nenhuma relação de agregação ou composição entre elas, chega-se a um cálculo de 57 pontos de função não ajustados.

Note que, se *Editora* e *Autor* não fossem considerados classes (arquivos lógicos), mas meramente campos a serem adicionados à classe *Livro*, a complexidade das funções relacionadas com Livro seria menor, pois haveria menos AL envolvidas. De outro lado, a consideração dessas informações como classes leva à conclusão de que os requisitos *ainda não estão completos*, pois as classes *Editora* e *Autor* deveriam ser consideradas cadastros e, portanto, funções de dados com suas respectivas operações CRUDL. Assim, após essa revisão nos requisitos, o número de pontos de função não ajustado deveria ser recalculado. Esse cálculo fica como sugestão de exercício para o leitor.

7.1.3 CONTAGEM INDICATIVA E ESTIMADA

Nem sempre, quando se vai fazer uma contagem de pontos de função, se tem à disposição informação detalhada sobre todas as funções de dados e transacionais, de forma que nem sempre a contagem detalhada é viável.

Assim, na falta dessas informações detalhadas, pode-se trabalhar com a *contagem indicativa* ou a *contagem estimada*. As técnicas apresentadas aqui seguem as definições da NESMA, conforme apresentado por Bomfim e Andrade (2015).

Evidentemente, que contagens paramétricas baseadas em informações menos detalhadas tendem a ser menos precisas. Estudos realizados pela NESMA indicam que a diferença entre a

contagem indicativa e a detalhada pode chegar a 50% nos piores casos. Mas, como foi salientado, na falta de maiores informações, é preferível ter alguma estimativa que depois possa ser ajustada do que não ter estimativa nenhuma.

A *contagem indicativa*, então, utiliza a penas o modelo conceitual como base para a estimação indicativa. Deve-se verificar apenas quantos ALI e AIE existem. A contagem indicativa pressupõe então que existam me média três EE (incluir, alterar e excluir), duas SE e uma CE para cada ALI e uma SE e uma CE para cada AIE. Assim, o número total de pontos de função não ajustados pode ser obtido simplesmente aplicando-se a fórmula:

$$UPF = 35 \times \#ALI + 15 \times \#AIE$$

Nesta fórmula, *#ALI* é o número de ALI e *#AIE* é o número de AIE presentes no modelo conceitual. No exemplo da seção anterior, como foram identificadas apenas duas transações ALI, o número de pontos de função na contagem indicativa seria 35 × 2 = 70, ou seja, 18,5% acima dos 57 pontos obtidos com a contagem detalhada. Mas considere que isto é apenas um pequeno exemplo no qual o número de transações é reduzidíssimo se comparado com aplicações reais nas quais é possível que as duas contagens sejam mais semelhantes.

A *contagem estimada* pode ser usada quando se tem um pouco mais de informação sobre as funções transacionais do sistema. Em especial, ela pode ser usada quando se conhece a lista das funções transacionais e de dados, mas não é possível ainda estimar sua complexidade.

Assim, todas as funções transacionais devem ser avaliadas como de complexidade média e todas as funções de dados como de complexidade baixa. Desta forma, considerando agora *#EE*, *#SE* e *#CE* como o número de entradas, saídas e consultas externas, o número de pontos de função não ajustados pode ser calculado como:

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

No exemplo da seção anterior, tem-se sete EE, um SE, quatro CE, dois ALI e nenhum AIE. Assim, a contagem estimada resulta em 7×4 + 1×5 + 4×4 + 2×7 + 0×5 = 28+5+16+14+0 = 63. Assim, esta contagem fica apenas 10,5% acima da contagem detalhada. Novamente vale o comentário de que o exemplo é muito pequeno para que se possa tirar alguma conclusão estatística sobre os métodos de contagem. Trata-se apenas de um exemplo para mostrar como a contagem é feita. Porém, mesmo assim, é possível observar que os resultados pelo menos não são muito discrepantes, mesmo para um exemplo pequeno.

Desta forma, pode-se optar por um dos três tipos de contagem dependendo de quanta informação se tenha disponível sobre o sistema, especialmente no caso de sistemas que ainda vão ser desenvolvidos. Mas, deve-se, sempre que possível, optar pela contagem mais detalhada, já que ela tende a ser mais precisa.

7.1.4 AFP – Pontos de Função Ajustados

O Governo Brasileiro não utiliza os fatores de ajuste de pontos de função propostos pelo IFPUG em seus projetos. Esses fatores existem porque dependendo do tipo de sistema, poderá ser mais difícil ou mais fácil desenvolver cada ponto de função. Sendo assim, poucas vezes esse tipo de análise acaba sendo utilizado no Brasil. Porém, caso a empresa desenvolvedora tenha interesse em saber de forma mais precisa se o projeto a ser desenvolvido é mais complexo ou mais simples que um projeto médio, poderá fazer o cálculo dos fatores de ajuste para descobrir se estes indicam um valor menor do que 1.0 (projeto mais simples) ou maior do que 1.0 (projeto mais complexo).

Por exemplo, um simples *front-end* para banco de dados terá suas funções desenvolvidas muito mais rapidamente do que um software para controle de um sistema de distribuição de energia elétrica. Porém, a funcionalidade dada por UFP, ou pontos de função não ajustados, não leva em

consideração a complexidade técnica interna das funções, mas apenas a percepção que um usuário tem delas. Para chegar a valores de esforço de desenvolvimento mais realistas, portanto, será necessário ajustar esse valor a partir dos fatores técnicos.

A técnica de pontos de função sugere quatorze fatores de ajuste técnico, conhecidos como GSC (*General Systems Characteristics*). Todos têm o mesmo peso, e a eles deve ser atribuída uma nota de 0 a 5, em que 0 significa que o fator não tem nenhuma influência no projeto e 5 significa que o fator tem influência determinante no projeto, sendo os valores de 1 a 4 intermediários.

Os quatorze GSC são os seguintes (mais detalhes na Seção 7.1.6):

- Comunicação de dados.
- Processamento distribuído.
- *Performance*.
- Configuração do equipamento.
- Volume de transações.
- Entrada de dados *on-line*.
- Interface com o usuário.
- Atualização *on-line*.
- Processamento complexo.
- Reusabilidade.
- Facilidade de implantação.
- Facilidade operacional.
- Múltiplos locais.
- Facilidade de mudanças (flexibilidade).

A avaliação dos GSC é feita para o projeto como um todo (não para cada função). Então, como são quatorze GSC e cada um receberá uma nota de 0 a 5, o somatório das notas ficará entre 0 e 70. Esse valor ajustado é conhecido como *TDI* (grau total de influência):

$$TDI = 0{,}65 + (0{,}01 \times GSC)$$

Assim, o somatório dos GSC, multiplicado por 0,01 e somado a 0,65, vai fazer que TDI varie de 0,65 a 1,35. Esse valor é multiplicado pelo número de UFP para obter o AFP, ou número de pontos de função ajustados:

$$AFP = UFP \times TDI$$

Em um projeto no qual todos os fatores técnicos sejam mínimos (nota 0), o AFP será igual a 65% do valor nominal de UFP. Já em um sistema em que todos os fatores técnicos sejam máximos (nota 5) o AFP será igual a 135% do valor nominal de UFP. Um sistema nominal seria aquele no qual todos os fatores técnicos têm nota 3, o que levaria o AFP a ser igual ao UFP.

7.1.5 Duração e Custo de um Projeto

Uma vez que o AFP do projeto tenha sido calculado, o esforço total será calculado multiplicando-se o AFP pelo índice de produtividade (*IP*) da equipe. Esse índice deve ser calculado para o ambiente local e pode variar muito em função do ambiente de trabalho, da experiência da equipe e de outros fatores.

Quando não é feito o ajuste de fatores técnicos, o esforço total é calculado diretamente sobre o UFP. Neste caso, deve-se considerar que diferentes linguagens de programação e diferentes tipos de projeto poderão determinar diferentes índices de produtividade. Assim, o esforço total do projeto é calculado como:

$$E = AFP \times IP$$

O *IP* de uma equipe pode ser calculado da seguinte forma: toma-se um projeto já desenvolvido, para o qual se saiba quanto esforço foi despendido. O esforço *E* pode ser contado em *desenvolvedor-mês, -semana, -dia,* ou *-hora*. O SERPRO utiliza a medida de *desenvolvedor-hora*.

Assim, se quatro pessoas trabalharam em tempo integral durante uma semana de quarenta horas, então o valor do esforço total desta equipe é de 160 desenvolvedores-hora. Para saber o índice de produtividade *IP* individual médio dessa equipe, precisamos saber quantos pontos de função eles desenvolveram nesta semana. Se a equipe desenvolveu, por exemplo, três relatórios (SE) de complexidade média, serão 3 × 5 = 15 pontos de função desenvolvidos na semana. Assim, o *IP* em horas por ponto de função é calculado como 160 / 15 = 10,7. Assim, essa equipe teve uma produtividade de 10,7 horas por ponto de função por desenvolvedor e seu *IP* é igual a 10,7 desenvolvedores-hora por ponto de função.

Se houver mais de um projeto disponível para cálculo, podem-se somar os *AFP* ou *UFP* de todos os projetos e dividir pelo esforço despendido em todos os projetos, obtendo sua média aritmética.

O SERPRO (Hazan, 2010) apresenta como referência índices de produtividade considerados baixos, médios e altos para projetos com diferentes linguagens de programação. Da tabela original, bastante extensa, destacamos na Tabela 7.6 os valores para as oito linguagens mais populares atualmente (outubro de 2018) segundo a tabela TIOBE (https://www.tiobe.com/tiobe-index/).

Tabela 7.6 Índices de produtividade esperados conforme a linguagem de programação (Fonte: Hazan, 2010)

Linguagem	**IP (horas por ponto de função)**		
	Baixa	**Média**	**Alta**
Java	14	10	6
C	24	18	12
C++	18	12	6
Python	18	14	8
Visual Basic	12	8	6
C#	17	12	7
PHP	15	10	5
JavaScript	16	12	8

Já o *custo* do projeto é calculado como o esforço total multiplicado pelo custo médio da hora do desenvolvedor e ambiente:

$$Custo = E \times Custo_{hora}$$

Há um site (vide *QR code*) [QRC 7.3] que armazena editais brasileiros de contratação de software em que a medida de custo é o ponto de função. No *site*, pode-se consultar o órgão público que lançou o edital, o ano, o objeto do projeto, o número total de UFP e o valor estimado. Desta forma é possível saber para diferentes áreas e diferentes projetos quanto custou o ponto de função. O valor varia bastante, mas em geral o custo fica entre cem e mil reais por ponto de função.

O *tempo linear ideal* para desenvolvimento do projeto pode ser calculado de forma simplificada. Inicialmente deve-se converter o esforço, se estiver em desenvolvedor-hora para desenvolvedor-mês. Isso pode ser feito dividindo-se o esforço *E* por 160 (número aproximado de horas trabalhadas em um mês). Um projeto, por exemplo, de baixa complexidade (espera-se alta

produtividade), com 10.000 pontos de função, a ser desenvolvido por uma equipe que usará a linguagem Java deverá, segundo a Tabela 7.6, trabalhar com produtividade de seis horas por ponto de função e, portanto, seriam 60.000 desenvolvedores-hora. O esforço em desenvolvedor-mês pode, então ser calculado como $E_{mês} = E / 160 = 60.000 / 160 = 375$. Assim, o esforço total do projeto será de 375 desenvolvedores-mês.

Para chegar no tempo calendário ou tempo linear ideal para desenvolver o projeto, pode-se aplicar uma equação simples que consiste em obter a raiz cúbica do esforço (*obrigatoriamente* em desenvolvedores-mês) multiplicado por 2,5. Aplicando-se este cálculo ao projeto com 375 desenvolvedores-mês chega-se a $2{,}5 \times \sqrt[3]{375}$, ou seja, aproximadamente 18 meses. Assim, a duração ideal do projeto seria de aproximadamente um ano e meio.

Para essa conta funcionar, entretanto, é necessário que o esforço *E* seja expresso em *desenvolvedor-mês*. O uso de outras unidades, como desenvolvedor-semana ou desenvolvedor-hora, vai provocar distorções no resultado devido ao uso da raiz cúbica.

Finalmente, o tamanho médio da equipe, ao longo de todo o desenvolvimento do projeto, será:

$$P = E / T$$

No exemplo, o tamanho médio da equipe seria dado por 375 / 18, ou seja, aproximadamente 21 pessoas. Porém, na prática, o tamanho da equipe poderá variar pois, como visto no Capítulo 6, usualmente projetos necessitam de menos pessoas nas suas fases iniciais e mais nas fases finais, especialmente durante a fase de Construção, caso se esteja usando o Processo Unificado como referência.

Além do tempo linear ideal, pode-se também falar em tempo mínimo de projeto, caso exista urgência para desenvolvê-lo e seja possível gastar mais recursos para tentar fazer isso num prazo mais curto. O tempo mínimo de um projeto pode ser calculado da seguinte forma:

$$T_{min} = 0{,}75 \times T$$

Assim, um projeto cujo tempo ideal é de 18 meses poderia ser acelerado até um tempo mínimo de 13,5 meses. Menos do que isso pode ser impossível.

Mas, se o projeto for desenvolvido em um tempo menor do que o ideal, espera-se que o esforço total seja maior do que o que foi inicialmente calculado. Assim, o custo total do projeto será mais caro e o tamanho médio da equipe também terá que ser maior do que E / T_{min}. Deve-se ter em mente que um projeto nessas condições estará sendo gerenciado em condições limite e, portanto, a princípio, apenas equipes muito experientes e organizadas teriam capacidade para dar conta desse ritmo de desenvolvimento. O modelo CII, apresentado na Seção 7.4 indica que uma redução de 25% no tempo linear de um projeto pode aumentar o seu esforço (*E*) total em 43%.

Todavia, de certa forma todas essas fórmulas são "mágicas", em razão da grande variedade de projetos e equipes de desenvolvimento de software. Antes de usá-las seriamente em uma empresa, convém sempre testá-las e ajustá-las para a realidade local.

7.1.6 Detalhamento dos Fatores Técnicos

Nesta seção será detalhada a interpretação dos fatores técnicos que compõem o TDI (Longstreet, 2018 – ver *QR code*) para que notas consistentes possam ser atribuídas [QRC 7.4]. Inicialmente, cada fator é descrito, e no final da seção uma tabela de referência (Tabela 7.7 adiante) para atribuição de notas para cada fator é apresentada.

O GSC *comunicação de dados* avalia o grau em que necessidades especiais de comunicação afetam o sistema. Sistemas isolados estariam no extremo inferior de avaliação; já os sistemas que fazem uso intensivo de dados obtidos em outros lugares e que enviam informação para muitos

lugares diferentes, através de diferentes protocolos de comunicação, estariam no extremo oposto.

O fator *processamento distribuído* avalia o grau em que dados distribuídos são usados pela aplicação. No extremo inferior estão os sistemas que armazenam os dados todos no mesmo lugar ou que, havendo necessidade de transferir dados, nada fazem para automatizar essa transferência. No extremo superior estão as aplicações que distribuem o processamento dinamicamente entre diferentes nodos, escolhendo sempre o melhor nodo possível para efetuar o processamento específico.

O fator *performance* avalia o grau em que a eficiência do sistema precisa ser considerada em sua construção. Sistemas eficientes sempre são desejáveis, mas esse fator avalia o quanto a eficiência é crítica para o sistema, de forma que se invistam recursos de tempo e dinheiro para melhorar esse aspecto. Um sistema de agendamento de compromissos, por exemplo, poderia estar no extremo inferior, porque normalmente trabalha com quantidades de dados pequenas e algoritmos simples, de forma que a eficiência não será uma preocupação. No extremo superior poder-se-ia ter sistemas que trabalham com quantidades absurdamente grandes de dados e/ou precisam apresentar respostas muito rápidas, como um sistema que interpreta dados de geoprocessamento em tempo real ou um sistema de direção automática de veículos.

O fator *configuração do equipamento* avalia o grau em que o sistema necessita ser projetado para compartilhar recursos de processamento. No extremo inferior estão as aplicações nas quais não há nenhuma preocupação com o compartilhamento de hardware; no extremo superior estão as aplicações nas quais o hardware estará sendo compartilhado, enquanto existem restrições operacionais da aplicação em si.

O fator *volume de transações* avalia a quantidade de transações simultâneas esperada. Os limites inferior e superior podem variar em função do desenvolvimento tecnológico do hardware e das redes de comunicação. O que há dez anos poderia ter sido considerado difícil em termos de taxa de transações hoje pode ser trivial. Então, é necessário que se avalie, em termos da tecnologia atual, qual seria a taxa de influência desse fator no desenvolvimento do produto. No limite inferior estariam sistemas com tão poucas transações que a tecnologia atual daria conta delas sem nenhuma preocupação extra. No extremo superior estariam sistemas no limite da tecnologia, os quais poderiam precisar de vários contêineres de processadores servidores para poder atender aos usuários (na casa de possíveis milhões de acessos simultâneos). Exemplos de sistemas com essa taxa de acesso são o Facebook® e o Google®.

O fator *entrada de dados on-line* avalia a porcentagem de informação que o sistema deve obter *on-line*, ou seja, dos usuários em tempo real. No extremo inferior estão os sistemas que tomam toda a informação necessária de arquivos ou repositórios. No extremo superior estão os sistemas nos quais a maioria das transações são entradas de dados *on-line*.

O fator *interface com o usuário* avalia o grau em que a aplicação será projetada para melhorar a eficiência do usuário final. Não ter preocupação nenhuma com isso leva o sistema ao limite inferior desse fator. Uma preocupação extrema em melhorar a eficiência do trabalho do usuário leva ao limite superior.

O fator *atualização online* avalia o percentual de arquivos internos que podem ser atualizados *on-line*. Se nenhum arquivo interno pode ser atualizado dessa forma, então deve-se atribuir nota mínima a esse fator. Quando todos os principais arquivos podem ser atualizados dessa forma e existem restrições especiais de volume e de segurança, então se está no extremo superior do fator.

O fator *processamento complexo* avalia o grau em que a aplicação utiliza processamento lógico ou matemático complexo. No extremo inferior estariam sistemas que apenas armazenam dados e eventualmente fazem algum tipo de soma ou produto; no outro extremo estariam sistemas baseados

em inteligência artificial e processamento numérico complexo nas áreas de Física e Engenharia.

O fator *reusabilidade* avalia em que grau a aplicação é projetada para ser reusável. No extremo inferior estão aplicações que são desenvolvidas sem nenhuma preocupação com reusabilidade e no extremo superior estão aplicações desenvolvidas para gerar vários componentes parametrizados reusáveis.

O fator *facilidade de instalação* avalia em que grau haverá preocupação em facilitar a instalação do sistema e a conversão dos dados. Um sistema para o qual não haja necessidade de conversão de dados e que deva ser instalado sem nenhuma automatização estará no limite inferior. Um sistema que deva se instalar automaticamente e converter automaticamente dados de sistemas legados estará no limite superior.

O fator *facilidade operacional* avalia em que grau a aplicação deverá fazer automaticamente processos de *startup*, recuperação de falhas e *backup*. Aplicações que não automatizem nada disso estão no limite inferior, e aplicações que automatizem totalmente os processos de operação estão no limite superior.

O fator *múltiplos locais* avalia o grau em que a aplicação é projetada para funcionar de forma distribuída. No limite inferior estão as aplicações isoladas *stand-alone* e no superior estão as aplicações formadas por diferentes módulos que rodam em máquinas geograficamente distribuídas ou dispositivos móveis.

O fator *facilidade de mudanças* (*flexibilidade*) avalia o grau em que a aplicação é projetada para facilitar mudanças lógicas e estruturais. Aplicações desenvolvidas sem esse tipo de preocupação estão no limite inferior, e aplicações projetadas para acomodar extensivamente mudanças futuras estão no limite superior.

A Tabela 7.7 apresenta as referências para aplicação das notas (0-5) de cada um dos quatorze fatores de juste técnico.

Tabela 7.7 Atribuição de notas aos fatores de ajuste técnico (Fonte: Longstreet, 2018)

GSC	Nota	Característica que melhor define o sistema
Comunicação de dados	0	Aplicações que são somente processamento em *batch* ou que rodam isoladas em um PC.
	1	Aplicações em *batch*, mas com entrada de dados remota *ou* saída remota, por exemplo, pela Web.
	2	Aplicações em *batch* com entrada de dados remota *e* saída remota.
	3	Aplicações que incluem coleta de dados *on-line* ou *front-end* de teleprocessamento para um sistema em *batch* ou sistema de consultas.
	4	Aplicações que são mais do que um *front-end*, mas suportam um único tipo de protocolo de comunicação.
	5	Aplicações que são mais do que um *front-end* e suportam vários tipos de protocolos de comunicação.
Processamento distribuído	0	Aplicações que não ajudam na transferência de dados ou funções de processamento entre os componentes do sistema.
	1	Aplicações que preparam os dados para o processamento do usuário final em outro componente do sistema, tal como sistemas que geram dados para serem lidos em uma planilha ou arquivo de processador de texto.
	2	Aplicações que preparam dados para transferência e então transferem e processam os dados em outro componente do sistema (não para processamento do usuário final).
	3	Aplicações em que o processamento distribuído e a transferência de dados ocorrem *on-line* e apenas em uma direção.
	4	Aplicações em que o processamento distribuído e a transferência de dados ocorrem *on-line* e nas duas direções.
	5	Aplicações em que as funções são executadas dinamicamente no componente mais apropriado do sistema.
Performance	0	Nenhum requisito de *performance* especial foi definido pelo cliente.
	1	Requisitos de *performance* foram estabelecidos e revisados, mas nenhuma ação especial precisa ser realizada.
	2	O tempo de resposta e a taxa de transferência são críticos durante as horas de pico. Nenhum *design* especial para utilização de CPU é necessário. O prazo para a maioria dos processamentos é o dia seguinte.
	3	O tempo de resposta e a taxa de transferência são críticos durante o horário comercial. Nenhum *design* especial para utilização de CPU é necessário. Os requisitos de prazo de processamento com sistemas interfaceados são restritivos.
	4	Em adição, os requisitos de *performance* são suficientemente restritivos para que se necessite estabelecer tarefas de análise de *performance* durante a fase de *design*.
	5	Em adição, ferramentas de análise de *performance* devem ser usadas nas fases de *design*, desenvolvimento e/ou implementação para atender aos requisitos de *performance* do cliente.

Configuração do equipamento	0	Nenhuma restrição operacional implícita ou explícita é incluída.
	1	Restrições operacionais existem, mas são menos restritivas do que em uma aplicação típica. Nenhum esforço especial é necessário para satisfazer as restrições.
	2	São incluídas algumas considerações sobre tempo e segurança.
	3	Requisitos específicos de processador para uma parte específica da aplicação são incluídos.
	4	Restrições sobre operações estabelecidas requerem que a aplicação tenha um processador dedicado ou prioridade de tempo no processador central.
	5	Em adição, existem restrições especiais na aplicação em relação aos componentes distribuídos do sistema.
Volume de transações	0	Não são antecipados períodos de picos de transações.
	1	Períodos de picos de transações (por exemplo, mensal, semestral ou anualmente) são antecipados.
	2	Picos de transação semanais são antecipados.
	3	Picos de transação diários são antecipados.
	4	Altas taxas de transação são estabelecidas pelo cliente nos requisitos da aplicação ou nos acordos de nível de serviço, as quais são suficientemente altas para necessitar de atividades de análise de *performance* na fase de *design*.
	5	Em adição, requer-se o uso de ferramentas de análise de *performance* nas fases de *design*, desenvolvimento e/ou instalação.
Entrada de dados online	0	Todas as transações são processadas em modo *batch*.
	1	1 a 7% das transações são entradas de dados interativas.
	2	8 a 15% das transações são entradas de dados interativas.
	3	16 a 23% das transações são entradas de dados interativas.
	4	24 a 30% das transações são entradas de dados interativas.
	5	Mais de 30% das transações são entradas de dados interativas.
Interface com o usuário		As notas são atribuídas a partir de uma lista de itens que podem ser considerados ou não: • Ajuda navegacional (por exemplo, teclas de função, menus gerados dinamicamente etc.). • Menus. • Ajuda e documentação on-line. • Movimentação de cursor automatizada. • Scrolling. • Impressão remota (a partir de transações on-line). • Teclas de função predefinidas. • Tarefas em batch submetidas a partir de transações on-line. • Seleção por cursor na tela de dados. • Alto uso de cores e destaque visual em tela. • Cópias impressas de documentação de usuário de transações on-line. • Interface por mouse. • Janelas pop-up. • Minimização do número de janelas para realizar objetivos de negócio. • Suporte bilíngue (conta como quatro itens). • Suporte para mais de duas línguas (conta como seis itens).
	0	Nenhuma das opções anteriores.
	1	De uma a três das opções anteriores.
	2	De quatro a cinco das opções anteriores.
	3	Seis ou mais das opções anteriores, mas não há requisitos específicos relacionados à eficiência de usuário final.
	4	Seis ou mais das opções anteriores e requisitos estabelecidos para a eficiência de usuário final são suficientemente fortes para requerer a inclusão de atividades de *design* para fatores humanos (por exemplo, minimizar a quantidade de cliques e movimentos de mouse, maximização de *defaults* e uso de *templates*).
	5	Seis ou mais das opções anteriores e requisitos estabelecidos para a eficiência de usuário são suficientemente fortes para requerer o uso de ferramentas e processos especiais para demonstrar que os objetivos foram atingidos.
Atualização on-line	0	Nenhuma atualização *on-line*.
	1	É incluída a atualização *on-line* para um a três arquivos. O volume de atualização é baixo e a recuperação é simples.
	2	A atualização *on-line* de quatro ou mais arquivos é incluída. O volume de atualização é baixo e a recuperação é simples.
	3	A atualização *on-line* dos principais arquivos lógicos internos é incluída.
	4	Em adição, proteção contra a perda de dados é essencial, e o sistema deve ser especialmente projetado para isso.
	5	Em adição, altos volumes de atualização trazem considerações de custo para o processo de recuperação. Procedimentos de recuperação altamente automatizados com intervenção mínima do operador são incluídos.
Processamento complexo		Os seguintes componentes são considerados para avaliação da complexidade do processamento da aplicação: • Controle cuidadoso (por exemplo, processamento especial de auditoria) e/ou processamento seguro específico da aplicação. • Processamento lógico extensivo. • Processamento matemático extensivo. • Muito processamento de exceções resultantes de transações incompletas que precisam ser reprocessadas, como transações de caixa-automático incompletas, causadas pela interrupção do teleprocessamento, valores de dados que faltam ou edições que falharam.

		• Processamento complexo para gerenciar múltiplas possibilidades de entrada e saída, como a multimídia ou a independência de dispositivos.
	0	Nenhuma das opções anteriores.
	1	Qualquer uma das opções anteriores.
	2	Quaisquer duas das opções anteriores.
	3	Quaisquer três das opções anteriores.
	4	Quaisquer quatro das opções anteriores.
	5	Todas as cinco opções anteriores.
Reusabilidade	0	Não há nenhuma preocupação para produzir código reusável.
	1	Código reusável é gerado para uso dentro da própria aplicação.
	2	Menos de 10% da aplicação deve considerar mais do que simplesmente as necessidades do usuário.
	3	10% ou mais da aplicação deve considerar mais do que as necessidades do usuário.
	4	A aplicação deve ser especificamente empacotada e/ou documentada para facilitar o reuso, e a aplicação deve ser personalizável pelo usuário em nível de código-fonte.
	5	A aplicação deve ser especificamente empacotada e/ou documentada para facilitar o reuso, e a aplicação deve ser personalizável por meio de manutenção de usuário baseada em parâmetros.
Facilidade de instalação	0	Nenhuma consideração especial foi estabelecida pelo usuário, e nenhum *setup* especial é necessário para a instalação.
	1	Nenhuma consideração especial foi estabelecida pelo usuário, mas um *setup* especial é requerido para a instalação.
	2	Requisitos de conversão e instalação foram estabelecidos pelo usuário, e guias de conversão e instalação devem ser fornecidos e testados. O impacto da conversão no projeto não é considerado importante.
	3	Requisitos de conversão e instalação foram estabelecidos pelo usuário, e guias de conversão e instalação devem ser fornecidos e testados. O impacto da conversão no projeto é considerado importante.
	4	Em adição à nota 2, ferramentas de conversão e instalação automática devem ser fornecidas e testadas.
	5	Em adição à nota 3, ferramentas de conversão e instalação automática devem ser fornecidas e testadas.
Facilidade operacional	0	Nenhuma consideração operacional especial, além dos procedimentos normais de *backup*, foi estabelecida pelo usuário.
	1-4	Um, alguns ou todos os itens a seguir se aplicam ao sistema (devem-se selecionar todos os que se aplicam. Cada item vale um ponto, exceto se for dito o contrário): • Processos efetivos de inicialização, backup e recuperação devem ser fornecidos, mas a intervenção do operador é necessária. • Processos efetivos de inicialização, backup e recuperação devem ser fornecidos, e nenhuma intervenção do operador é necessária (conta como dois itens). • A aplicação deve minimizar a necessidade de armazenamento em fitas (ou qualquer outro meio de armazenamento off-line). • A aplicação deve minimizar a necessidade de manuseio de papel.
	5	A aplicação é projetada para operar de forma não supervisionada. "Não supervisionada" significa que não é necessária nenhuma intervenção do operador do sistema, a não ser, talvez, na sua primeira inicialização ou desligamento final. Uma das características da aplicação é a recuperação automática de erros.
Múltiplos locais	0	Os requisitos do usuário não exigem a consideração da necessidade de mais do que um usuário ou instalação.
	1	A necessidade de múltiplos locais deve ser considerada no projeto, e a aplicação deve ser projetada para operar apenas em ambientes idênticos de hardware e software.
	2	A necessidade de múltiplos locais deve ser considerada no projeto, e a aplicação deve ser projetada para operar apenas em ambientes de hardware e software similares.
	3	A necessidade de múltiplos locais deve ser considerada no projeto, e a aplicação é projetada para operar em ambientes de hardware e software diferentes.
	4	O plano de documentação e suporte deve ser fornecido e testado para suportar a aplicação em múltiplos locais, e a aplicação é como descrita nas notas 1 e 2.
	5	O plano de documentação e suporte deve ser fornecido e testado para suportar a aplicação em múltiplos locais, e a aplicação é como descrita na nota 3.
Facilidade de mudanças (flexibilidade)		As seguintes características podem se aplicar ao software: • Facilidades de consulta e relatório flexíveis devem ser fornecidas para tratar consultas simples, por exemplo, operadores lógicos binários aplicados apenas a um arquivo lógico interno (conta como um item). • Facilidades de consulta e relatório flexíveis devem ser fornecidas para tratar consultas de complexidade média, por exemplo, operadores lógicos binários aplicados a mais do que um arquivo lógico interno (conta com dois itens). • Facilidades de consulta e relatório flexíveis devem ser fornecidas para tratar consultas de complexidade alta, por exemplo, combinações de operadores lógicos binários em um ou mais arquivos lógicos internos (conta como três itens). • Dados de controle de negócio são mantidos em tabelas gerenciadas pelo usuário e com processos interativos on-line, mas as mudanças só têm efeito no dia seguinte (conta como um item). • Dados de controle de negócio são mantidos em tabelas gerenciadas pelo usuário e com processos interativos on-line, e as mudanças têm feito imediatamente (conta como dois itens).
	0	Nenhum item.
	1	Um item.

	2	Dois itens.
	3	Três itens.
	4	Quatro itens.
	5	Cinco itens ou mais.

Ressalta-se que mesmo que os fatores de ajuste não sejam usados em projetos medidos com UPF, seu cálculo pode ser muito interessante para indicar o quanto o projeto é mais complexo ou mais simples do que um projeto médio.

7.2 Pontos de Histórias

Pontos de histórias (PH), ou *story points*, é a estimativa de esforço preferida (embora não exclusiva) de métodos ágeis como *Scrum* e XP. Um ponto de história não é uma medida de complexidade funcional, como os pontos de função ou pontos de caso de uso, mas uma medida de esforço relativa à equipe de desenvolvimento.

Por um lado, a técnica se aproxima muito da estimação por especialista, já que a equipe ágil usará hipóteses qualitativas e não quantitativas para estimar o esforço de cada história de usuário. Porém, a técnica tem a vantagem de poder ser aplicada a praticamente qualquer tipo de sistema, enquanto que pontos de função se aplica mais facilmente a sistemas do tipo comercial. Sistemas de outros tipos, como jogos ou sistemas científicos, poderão apresentar dificuldades imensas para a aplicação de pontos de função. Por exemplo, no caso de um jogo em realidade virtual em primeira pessoa, no caso de uma biblioteca de funções matemáticas ou no caso de um compilador, quais são as funções de dados e funções transacionais?

Segundo Kniberg (2007), uma estimativa baseada em pontos de histórias deve ser feita pela equipe. Inicialmente, pergunta-se à equipe quanto tempo *X* pessoas que se dedicassem unicamente a uma história de usuário levariam para terminá-la, gerando uma versão executável funcional. Se a resposta for, por exemplo, "três pessoas levariam quatro dias", então atribua à história $3 \times 4 = 12$ pontos de história. Pode-se usar aqui a técnica de jogo de planejamento (*planning poker*), conforme já explicado no Capítulo 4.

Assim, um ponto de história pode ser definido como o esforço de desenvolvimento de uma pessoa durante um dia ideal de trabalho, que consiste em uma pessoa dedicada de seis a oito horas a um projeto, sem interrupções nem atividades paralelas.

7.2.1 Atribuição de Pontos de Histórias

Nos métodos ágeis, a importância da estimativa normalmente está na comparação entre histórias, ou seja, mais importante do que saber quantos dias uma história efetivamente levaria para ser implementada é saber que uma história levaria duas vezes mais tempo do que outra para ser implementada. Por esse motivo, os pontos de histórias são atribuídos normalmente não como valores da série dos números naturais, mas como valores da série aproximada de números de Fibonacci. Um número de Fibonacci é definido como a soma dos dois números de Fibonacci anteriores na série (com exceção dos dois primeiros, que, por definição, são 1 e 1). Assim, o início da série de Fibonacci é constituído pelos números: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 etc. Porém, pode ser estranho mensurar pontos de histórias em 89 ou 34 pontos. Então, na prática, faz-se uma aproximação desses valores para uma série como 1, 2, 3, 5, 8, 15, 25, 40, 60, 100 etc. A ideia é que os pontos de história apresentem uma ordem de grandeza natural para o esforço, e não uma medida exata.

Outra opção para estimar pontos de histórias é usar o sistema "camiseta", com valores "pequeno", "médio" e "grande".

O procedimento de atribuição de pontos funciona assim: tomam-se da lista de histórias de usuário previamente preparada aquelas consideradas mais simples e atribuem-se a elas 1 ou 2 pontos. Depois, sequencialmente, pegam-se outras mais complexas, inicialmente de 3 pontos, depois de 5, e assim por diante. O motivo é que, para o ser humano, é muito mais fácil fazer medidas relativas do que medidas absolutas. Assim, é difícil, por exemplo, uma pessoa estimar o peso de um cavalo sem ter uma balança ou conhecimento prévio do valor. Mas uma pessoa consegue verificar facilmente que um cavalo pesa menos do que um elefante e mais do que um cachorro.

Felix (2009) comenta que a atribuição de pontos de histórias usualmente segue critérios subjetivos de complexidade, esforço e risco, sendo caracterizada por frases como:

- *Complexidade*: "Essa regra de negócio tem muitos cenários possíveis".
- *Esforço*: "Essa alteração é simples, mas precisa ser realizada em muitas telas".
- *Risco*: "Precisamos utilizar o *framework* X, mas ninguém na equipe tem experiência".

Na maioria dos modelos ágeis, o esforço precisa ser estimado pela equipe como um todo, e não por um gerente, porque é medida justamente a capacidade das pessoas de realizar as tarefas. Por isso, é necessário que as pessoas que vão fazer o trabalho efetivamente possam opinar e avaliar a carga de trabalho, que, depois de decidida, passará a ser um compromisso de desenvolvimento.

7.2.2 MEDIÇÃO DE VELOCIDADE

Pontos de histórias são usados por equipes ágeis para medir sua velocidade de projeto. Pode-se fazer um gráfico e deixar à vista de todos onde, a cada iteração, são medidos os pontos de histórias efetivamente desenvolvidos. Se essa velocidade começar a cair, a equipe deverá verificar o motivo. Vários motivos podem ser listados: desmotivação, erros de estimação, erros de priorização (a equipe começou a tratar histórias de usuário mais simples e deixou as mais complexas e arriscadas para depois) etc.

Em todo caso, quando se deseja aumentar a velocidade da equipe, é comum que se faça a aquisição de mais desenvolvedores para trabalhar nas iterações restantes. Também é natural que, enquanto os novos desenvolvedores estiverem se ambientando, a velocidade em PH seja reduzida para mais tarde aumentar. A Figura 7.1 mostra um exemplo de gráfico de velocidade de projeto em que, após a aquisição de novos membros para a equipe, há uma redução na velocidade em PH que é compensada após três iterações, quando então os novos membros da equipe passam a ser efetivamente produtivos.

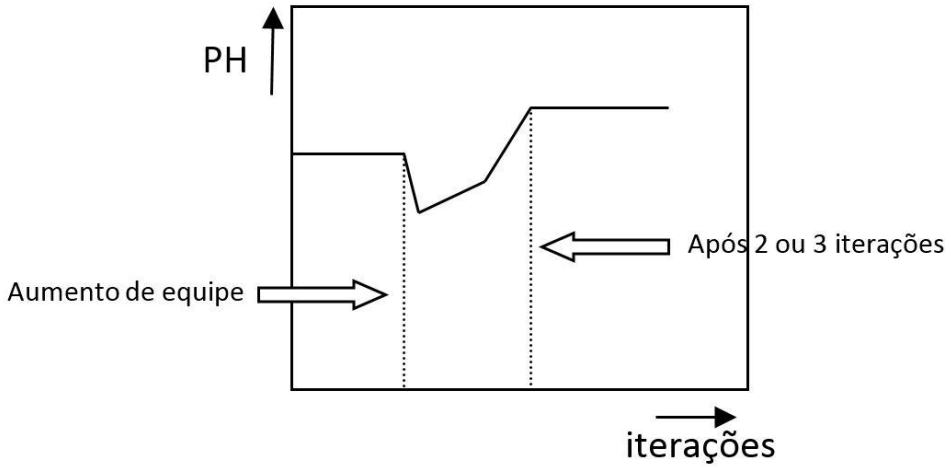

Figura 7.1 Gráfico de velocidade de projeto em pontos de histórias

Normalmente, um gráfico de velocidade é relativamente estável. Embora os valores possam variar de uma iteração para outra, sua derivada se mantém constante. Se houver medidas de melhoria de produtividade, pode-se esperar aumentos na velocidade, ou seja, uma derivada positiva. Então, a principal utilidade do gráfico de velocidade consiste em ajudar a diagnosticar possíveis problemas de ambiente, caso a derivada se torne negativa, bem como verificar a efetividade de melhorias no processo produtivo.

7.3 SLOC e KSLOC

A técnica de estimação conhecida como LOC (*Lines of Code*) ou SLOC (*Source Lines of Code*) foi possivelmente a primeira a surgir e consiste em estimar o número de linhas de código que um programa deverá ter, normalmente com base na opinião de especialistas e no histórico de projetos passados.

Esse tipo de técnica surgiu numa época em que as linguagens de programação, como FORTRAN, eram fortemente orientadas a linhas de código. Naquele tempo, programas eram registrados em cartões perfurados, uma linha de código por cartão, de forma que a altura da pilha de cartões era uma medida bastante natural para a complexidade de um programa.

Rapidamente, a técnica evoluiu para a forma conhecida como KSLOC (*Kilo Source Lines of Code*), tendo em vista que o tamanho da maioria dos programas passou a ser medido em milhares de linhas. Uma unidade KSLOC, portanto, vale mil unidades SLOC. Além disso, também são usados os termos MSLOC para milhões de linhas e GSLOC para bilhões de linhas de código.

Hoje em dia, as linguagens de programação são muito mais flexíveis em relação a linhas de código, permitindo que vários comandos ocorram em uma única linha ou ainda que um comando seja dividido em várias linhas. Mesmo assim, a noção de linha de código continua sendo uma medida popular para complexidade de programas, que podem variar de 10 a 100.000.000 de linhas, com complexidade inerente diretamente proporcional na maioria das vezes.

De fato, a medida faz sentido quando se comparam ordens de magnitude. Um programa com 100.000 linhas possivelmente será mais complexo do que um programa com 10.000 linhas. Mas pouco se pode concluir ao se comparar um programa com 10.000 linhas e um programa com 11.000 linhas.

7.3.1 Estimação de KSLOC

Uma técnica para estimação de KSLOC é reunir a equipe para discutir o sistema a ser desenvolvido. Cada participante dará a sua opinião sobre a quantidade de KSLOC que será necessária para desenvolver o sistema.

Como no caso da técnica de *poker planning*, pode-se sugerir que cada participante anote em um papel o número de sua estimativa, sem que os outros vejam. Depois, todos abrem suas estimativas. Se houver estimativas muito acima ou muito abaixo das demais, os autores devem defender seu ponto de vista e o processo é repetido até que se chegue próximo a um consenso.

Porém, nem sempre a equipe chega a um valor de estimativa consensual. Neste caso, sugere-se que a equipe trabalhe com uma média ponderada entre os valores máximo, mínimo e médio, usando:

- O KSLOC *otimista*, ou seja, a estimativa mais baixa.
- O KSLOC *pessimista*, ou seja, a estimativa mais alta.
- O KSLOC *médio*, ou seja, a média das estimativas dos demais participantes, excetuando a mais

alta e a mais baixa.

A partir desses valores, o KSLOC pode ser calculado assim:

$$KSLOC = (4 \times KSLOC_{médio} + KSLOC_{otimista} = KSLOC_{pessimista}) / 6$$

Essas estimativas devem ser comparadas com a informação real, ao final do projeto. Por isso, a equipe deve ter um *feedback* para ajustar futuras previsões. Por exemplo, um membro da equipe que costuma prever valores muito abaixo do real deverá perceber que está sendo muito otimista e tentar ajustar suas previsões para valores mais altos no futuro.

Uma técnica que pode ser empregada para ajustar a capacidade de previsão da equipe é tomar projetos já desenvolvidos, para os quais se saiba de antemão a quantidade de linhas, e exercitar a estimativa com a equipe, comparando mais tarde os valores reais com os valores estimados por seus membros.

Embora o objetivo seja acertar a previsão, isso na prática é quase impossível. Erros de previsão de 20% ou 30% não chegam a ser muito significativos, ou seja, é perfeitamente aceitável fazer uma previsão de 10.000 linhas para um projeto de 12.000 ou 13.000 linhas. Mas seria um problema caso a equipe fizesse uma previsão de 10.000 linhas para um projeto que, ao final, tivesse 30.000 ou 40.000 linhas (erro de 200 e 300%, respectivamente).

A medida de número de linhas, porém, ainda pode ser traiçoeira. Possivelmente, programadores experientes produzirão software com menos linhas do que programadores menos experientes, e diferentes linguagens de programação também apresentam diferentes valores de produtividade por programador ou por ponto de função. Então, alguns cuidados devem ser tomados. Não se deve considerar SLOC uma boa medida de produtividade individual, pois, muitas vezes, a complexidade inerente de um programa vai muito além da quantidade de linhas: poucas linhas de código poderão realizar funcionalidades difíceis e complexas, enquanto muitas linhas poderão efetuar apenas funcionalidades triviais. É o caso de quem desenvolve software científico ou de inteligência artificial em contraponto com a produção de meros cadastros e relatórios.

Outra situação a considerar é o fato de que refatorações do software poderão remover muitas linhas de código inútil ou redundante, e isso não deve ser considerado uma produtividade negativa.

Assim, a medida do número de linhas de código pode ser útil no longo prazo ou para um projeto como um todo, mas seu uso para avaliar atividades diárias dos desenvolvedores ou em pequenas partes de projeto é bastante arriscado.

7.3.2 Como Contar Linhas de Código

Dependendo da linguagem de programação, pode-se perguntar o que efetivamente conta como linha de código. Em primeiro lugar convém distinguir o que são linhas físicas e linhas lógicas. Linhas *físicas* são todas as linhas do arquivo de texto que contém o código fonte e estas, evidentemente são muito fáceis de contar. Mas a maioria dos programadores, até por necessidade de obedecer a padrões de estilo, deixam linhas em branco em seus programas e estas efetivamente não podem ser consideradas para efeito de estimação de esforço. Assim, apenas linhas *lógicas*, ou seja, aquelas que contém comandos ou declarações são consideradas, usualmente, quando se trata de calcular KSLOC.

Mas ainda assim ficariam algumas dúvidas: “else” conta? Declaração de variável conta? Park (1992) apresenta, para várias linguagens de programação, os tipos de comandos que devem ser efetivamente contados e quais não devem ser contados como SLOC. A Tabela 7.8 apresenta uma versão revisada e atualizada dessas orientações (Nguyen, Deeds-Rubin, Tan & Boehm, 2007), mencionando linguagens como Java, que não existiam ainda em 1992.

Tabela 7.8 Padrão para contagem de SLOC em Java, C, C++ e C#

Precedência	**Estrutura**	**Regra de contagem**
1	Comandos de seleção: *if*, *else if*, *else*, operador "?", *try*, *catch*, *switch*	Conta uma vez cada ocorrência. Comandos aninhados são contados de forma similar.
2	Comandos de iteração: *for*, *while*, *do..while*	Conta uma vez cada ocorrência. A inicialização, a condição e o incremento da instrução *for* não contam, bem como quaisquer outras expressões opcionais do *for*.
3	Comandos de desvio: *return*, *break*, *goto*, *exit*, *continue*, *throw*	Conta uma vez cada ocorrência. Rótulos usados com comandos *goto* não contam.
4	Expressões: Chamada de função, atribuição, comando vazio	Conta uma vez cada ocorrência.
5	Outras expressões: Expressões que terminam com ";"	Conta uma vez cada ocorrência. Expressões consistindo unicamente de ";" dentro de comandos de iteração não contam.
6	Delimitadores de bloco {...}	Contam uma vez por par exceto se o fechamento for sucedido por ";", ou seja, "};". Chaves usadas com comandos de seleção e iteração não contam. Definição de função que necessariamente inclui delimitadores de bloco conta uma única vez.
7	Diretivas de compilação	Conta uma vez cada ocorrência.
8	Declaração de dados	Conta uma vez cada ocorrência. Isso inclui protótipos de função, declaração de variáveis e declarações *typedef*. Palavras chave como *struct* e *class* não contam.

A Tabela 7.8 apresenta, então, um resumo dessas orientações, indicando como efetuar a contagem de programas nas linguagens Java, C, C++ e C#. Na tabela, cada estrutura deve ser classificada em uma única categoria (linha da tabela) seguindo a prioridade de cima para baixo, ou seja, se uma estrutura eventualmente pudesse ser classificada na linha quatro e também na linha cinco ela deve ser considerada como classificada na linha quatro, que tem precedência maior.

O trabalho de Nguyen, Deeds-Rubin, Tan e Boehm (2007) apresenta em seu anexo tabelas com regras semelhantes para as linguagens PERL, JavaScript e SQL, as quais podem ser consultadas no original com o uso do *QR code* ao lado [QRC 7.5].

Assim, um programa como o apresentado na Tabela 7.9, deve contabilizar oito SLOC.

Tabela 7.9 Exemplo de contagem de linhas de código lógicas em Java

	Contagem	Regra (precedência)
public class Fibonacci {	-	8
	-	
static long fibo(int n) {	1	6
if (n < 2) {	1	1
return n;	1	3
} else {	1	1
return fibo(n - 1) + fibo(n - 2);	1	3
}	-	
}	-	
	-	
public static void main(String[] args) {	1	6
	-	
// teste do programa. Imprime os 30 primeiros termos	-	
for (int i = 0; i < 30; i++) {	1	2
System.out.print("(" + i + "):" + Fibonacci.fibo(i) + "\t");	1	5
}	-	
}	-	
	-	
}	-	
TOTAL	**8**	

No caso de declaração de variáveis, se várias variáveis puderem ser declaradas em uma mesma expressão, no caso, se são todas do mesmo tipo, conta-se uma vez. Mas se são variáveis de tipo diferente, elas necessariamente têm que ser declaradas em expressões distintas e assim, conta-se uma vez para cada tipo de variável declarada.

7.3.3 Transformando Pontos de Função em KSLOC

Existem estudos (Jones C., 1996) que apresentam uma relação entre o número de pontos de função não ajustados e o número de linhas de código que se pode esperar em média, conforme a linguagem de programação. Os resultados comparativos são apresentados em tabelas conhecidas como *backfiring* ou *backfire tables*.

Uma versão mais recente e atualizada de *backfire table* é apresentada no site da organização QSM (Quantitative Software Management) [QRC 7.6], que em sua versão 5.0 apresenta os resultados obtidos a partir de mais de dois mil projetos executados com pontos de função linhas de código mensurados (ver *QR code*). Das 126 linguagens de programação analisadas, 37 apresentaram dados suficientes para que conclusões estatisticamente relevantes pudessem ser apresentadas. Destas, algumas são resumidas na Tabela 7.10.

Tabela 7.10 *Backfire table* (Fonte: http://www.qsm.com/resources/function-point-languages-table)

Linguagem	Média	Mediana	Melhor caso	Pior Caso
Assembly	119	98	25	320
C	97	99	39	333
C++	50	53	25	80
C#	54	59	29	70

Java	53	53	14	134
Perl	24	15	15	60
SQL	21	21	13	37
Visual Basic	42	44	20	60

Então, a tabela, indica para cada linguagem de programação e seus projetos típicos quantas linhas lógicas de código se pode esperar para cada ponto de função e vice-versa.

Assim, um sistema desenvolvido ou a ser desenvolvido em Java que, pela técnica de Análise de Pontos de Função, tem seu tamanho estimado em 1.000 pontos de função terá um tamanho em SLOC médio estimado em 53.000 linhas, sendo 14.000 no melhor caso e 134.000 no pior caso.

A técnica também pode ser usada para estimar o número de pontos de função equivalente de um sistema para o qual se conheça o número de KSLOC. Assim, por exemplo, um programa com 1.000.000 de linhas de código em Java deve em média implementar 1.000.000 / 53 = 18.868 pontos de função, ou 1.000.000 / 14 = 71.429 pontos de função no melhor caso, ou ainda 1.000.000 / 134 = 7.463 pontos de função no pior caso.

Certo cuidado tem que ser tomado ao se interpretar esta tabela. Ela não indica, por exemplo, que um sistema desenvolvido em C (média 97) terá metade das linhas de código se for desenvolvido em C++ (média 50). A tabela apresenta dados de projetos diferentes em cada linguagem. Assim, um projeto que é desenvolvido em SQL dificilmente também seria desenvolvido em Assembly e vice-versa.

7.4 COCOMO II

COCOMO (pronuncia-se "côcomo") ou *Constructive Cost Model* é um modelo de estimativa de esforço baseado em KSLOC. Sua primeira versão, conhecida como COCOMO 81, já é considerada obsoleta e foi substituída por COCOMO II, também conhecido como CII, em aplicações reais. A versão mais básica de COCOMO 81 é apresentada aqui porque sua simplicidade conceitual ajuda a esclarecer conceitos usados por outras técnicas. Além disso, ela pode ser usada como uma ferramenta de estimativa grosseira, caso a única informação disponível sobre o sistema seja o número de linhas de código.

O modelo COCOMO foi criado por Boehm (1981) a partir de um estudo empírico sobre 63 projetos na empresa TRW Aerospace. Os programas examinados tinham de duas a cem KSLOC e eram escritos em linguagens diversas, como Assembly e PL/I. Mais tarde o modelo foi ajustado a partir de dados coletados de outros 161 projetos.

7.4.1 Modelo Básico de COCOMO 81

A implementação mais simples de COCOMO é capaz de calcular esforço, tempo e tamanho de equipe a partir de uma simples estimativa de KSLOC. Para o cálculo do esforço com COCOMO 81 deve-se considerar o tipo de projeto a ser desenvolvido:

- *Modo orgânico*: aplica-se quando o sistema a ser desenvolvido não envolve dispositivos de hardware e a equipe está acostumada a desenvolver esse tipo de aplicação, ou seja, sistemas de baixo risco tecnológico e baixo risco de pessoal.
- *Modo semidestacado*: aplica-se a sistemas com maior grau de novidade para a equipe e que envolvem interações significativas com hardware, mas sobre os quais a equipe ainda tem algum conhecimento, ou seja, sistemas nos quais a combinação do risco tecnológico e de pessoal seja média.
- *Modo embutido*: aplica-se a sistemas com alto grau de interação e diferentes dispositivos de

hardware, ou que sejam embarcados, e para os quais a equipe tenha considerável dificuldade de abordagem. São os sistemas com alto risco tecnológico e/ou de pessoal.

Uma vez definido o modo, o modelo permite determinar as três informações básicas gerais sobre o sistema:

- O esforço estimado em desenvolvedor-mês: *E*.
- O tempo linear de desenvolvimento sugerido em meses corridos: *D*.
- O número médio de pessoas recomendado para a equipe: *P*.

O número de pessoas para a equipe será sempre dado por $P = E / D$. A medida parece simplista, porque aparentemente considera que a relação entre o número de pessoas e o tempo de projeto é linear. Mas o cálculo do esforço estimado (*E*) já leva em conta a não linearidade dessa relação, pois é uma função exponencial, como será visto adiante.

Além disso, é importante que nas fórmulas de COCOMO 81 e CII seja usada sempre a unidade *desenvolvedor-mês*. Variações como *desenvolvedor-semana*, *desenvolvedor-dia* ou *desenvolvedor-hora* poderão provocar distorções nos resultados, em razão do uso de exponenciais nas fórmulas.

Inicialmente, calcula-se o esforço (*E*) a partir da seguinte fórmula:

$$E = ab \times KSLOC^{bb}$$

Nesta fórmula, *ab* e *bb* são obtidos a partir da Tabela 7.11.

Tabela 7.11 Valores de *ab*, *bb*, *cd* e *db* em função do tipo de projeto

Tipo de projeto	***ab***	***bb***	***cb***	***db***
Orgânico	2,4	1,05	2,5	0,38
Semidestacado	3,0	1,12	2,5	0,35
Embutido	3,6	1,2	2,5	0,32

Já o tempo linear ideal recomendado para o desenvolvimento é dado por:

$$T = cb \times E^{db}$$

Nesta fórmula, *cb* e *db* são constantes também obtidas na Tabela 7.11.

Por exemplo, um projeto com KSLOC estimado em 20 (20.000 linhas de código) com baixo risco (modo orgânico) produzirá as seguintes estimativas:

$$E = 2{,}4 \times 20^{1{,}05} = 56 \text{ desenvolvedores-mês}$$

$$T = 2{,}5 \times 56^{0{,}38} = 11{,}5 \text{ meses}$$

$$P = 56 / 11{,}5 = 5 \text{ pessoas}$$

Todos os valores são aproximados em razão das casas decimais, mas a conclusão do modelo COCOMO básico é que um projeto orgânico com previsão de 20.000 linhas de código será desenvolvido em cerca de 1 ano por uma equipe de cerca de 5 pessoas em média.

O modelo básico é bom por ser simples e rápido, mas sua capacidade preditiva é limitada, em razão do fato de que o esforço de desenvolvimento não é função apenas do número de linhas de código, mas também de outros fatores, que são considerados no modelo CII.

7.4.2 Visão Geral de CII

COCOMO II ou CII é uma evolução do modelo COCOMO 81 e, ao contrário de seu antecessor, funciona bem com ciclos de vida iterativos, sendo fortemente adaptado para uso com o Processo Unificado (Boehm B., 2000), embora também seja definido para os modelos Cascata e Espiral.

O CII foi projetado para mensurar o esforço e o tamanho médio de equipe para as fases de elaboração e construção do Processo Unificado. Então, o esforço e a equipe para as fases de

concepção e transição podem ser calculados como uma fração dos valores obtidos para as duas outras fases. A Figura 7.2 apresenta esquematicamente a região de estimação de esforço abrangida pelo método CII.

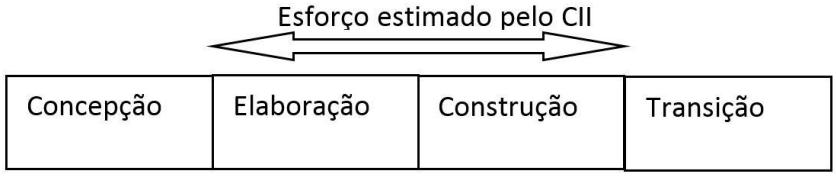

Figura 7.2 Região de estimação de esforço de CII

Como se pode ver na figura, CII é aplicado para determinar o esforço necessário para desenvolver as fases de elaboração e construção de um projeto. A duração das fases de concepção (caso ainda não tenha terminado) e transição deve ser calculada pela aplicação de um percentual sobre o esforço obtido para as fases de elaboração e construção, em média 5% para a concepção e 10% para a transição.

O método CII define o esforço total, em desenvolvedor-mês, a partir do número de KSLOC, de uma constante ajustável por dados históricos *A*, de um valor que pode ser calculado para cada projeto, chamado de *coeficiente de escala S*, e por um conjunto de fatores multiplicadores de esforço M_i, os quais também são individualmente calculados para cada projeto a partir de notas dadas. A equação geral que define o esforço total nominal de um projeto é a seguinte:

$$E = A \times KSLOC^S \times \prod_{i=1}^{n} Mi$$

Nesta fórmula:

- *E* é o esforço total nominal que se deseja calcular para o projeto (fases de elaboração e construção em desenvolvedores-mês).
- *A* é uma constante que deve ser calibrada a partir de dados históricos. CII sugere um valor inicial de 2,94.
- *KSLOC* é o número estimado de milhares de linhas de código que deverão ser desenvolvidas.
- *S* é o coeficiente de escala, cujo cálculo é explicado na Seção 7.4.3.
- M_i são os multiplicadores de esforço, que são explicados na Seção 7.4.4.

O valor *E* corresponde, então, ao esforço total em desenvolvedor-mês para as duas fases centrais do UP. Mas, normalmente, projetos são desenvolvidos por equipes e não por uma única pessoa. Assim, uma equipe maior pode conseguir desenvolver um projeto mais rápido do que uma equipe menor. Existe um limite a partir do qual a equipe será grande demais para o tamanho do projeto, e o esforço de gerenciar a equipe não compensará mais o ganho com tempo, havendo inclusive, uma inversão da curva no sentido de que quanto maior a equipe, maior será o tempo do projeto (Figura 7.3).

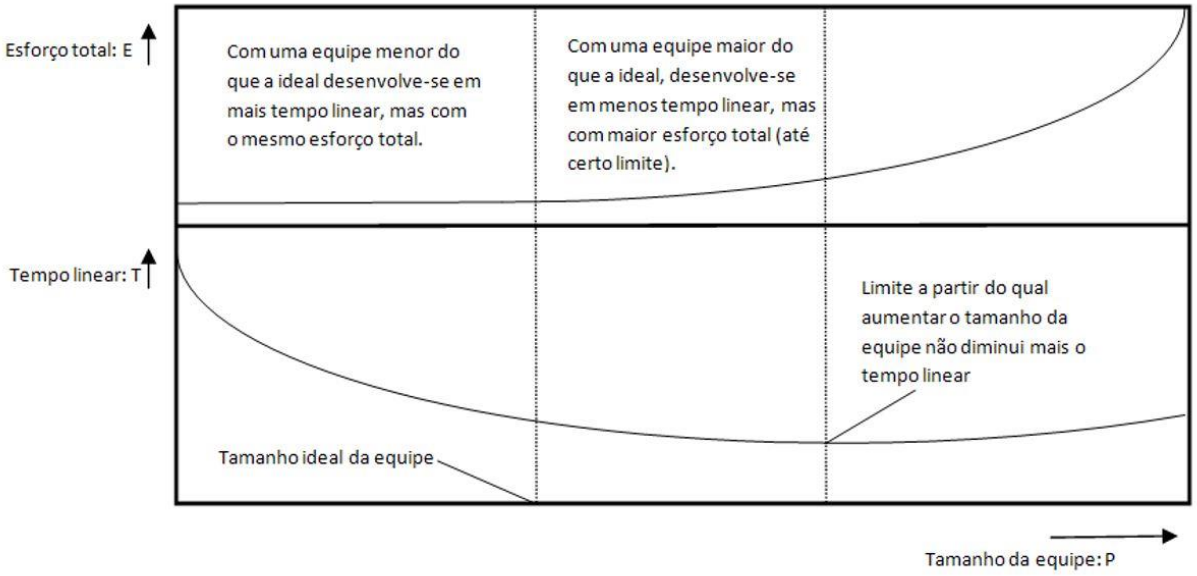

Figura 7.3 Relação entre o tamanho da equipe e o tempo linear de desenvolvimento de um projeto

Pode-se falar, assim, em um tempo ideal e um tamanho de equipe ideal para desenvolver um projeto. O CII sugere que o tempo linear ideal para desenvolver um projeto cujo esforço total *E* já é conhecido seja calculado a partir da seguinte fórmula:

$$T = C \times E^{D+0,2\times(S-B)}$$

Nesta fórmula:

- *T* é o tempo linear ideal de desenvolvimento.
- *B*, *C* e *D* são constantes que devem ser calibradas a partir de dados históricos.
- *E* é o esforço total para o projeto, conforme calculado anteriormente.
- *S* é o coeficiente de escala que já foi mencionado.

Finalmente, o *tamanho médio da equipe P* é obtido pela simples divisão do esforço total pelo tempo linear:

$$P = E / T$$

Essa fórmula, porém, só se aplica quando o tempo *T* for efetivamente o tempo linear ideal ou superior a este. De acordo com a Figura 7.3, pode-se perceber que uma redução no tempo linear não implica em um aumento de equipe proporcional, ou seja, para fazer o projeto na metade do tempo, não adianta dobrar o tamanho da equipe.

O método CII possui um multiplicador de esforço especificamente para indicar essa relação. O multiplicador SCED (cronograma de desenvolvimento requerido), apresentado na Seção 7.4.4, tem valor nominal igual a 1,0 se o projeto deve ser desenvolvido no seu tempo ideal. Caso se queira forçar um desenvolvimento mais rápido, em 85% do tempo ideal, SCED vai valer 1,14. Caso se queira forçar um desenvolvimento ainda mais rápido, em 75% do tempo ideal, SCED passa a valer 1,43, ou seja, indicando um esforço 43% maior para obter uma redução menor do que 25% no tempo linear. Isso porque o novo tempo linear será maior do que 25% do tempo originalmente calculado, já que o esforço total também será maior.

Os valores das constantes *A*, *B*, *C* e *D* foram obtidos pela equipe da USC (University of Southern

California) a partir da análise de 161 projetos. Tais valores são os seguintes:

- $A = 2{,}94$
- $B = 0{,}91$
- $C = 3{,}67$
- $D = 0{,}28$

Porém, recomenda-se que pelo menos o valor de *A* seja calibrado a partir de dados obtidos na organização local que estiver usando a técnica (Seção 7.4.6).

Um *site* mantido pela USC (*University of Southern California*) disponibiliza uma ferramenta *online* (ver *QR code*) que permite o cálculo automatizado do esforço baseado em CII usando como parâmetro tanto KSLOC quanto pontos de função. [QRC 7.7]

7.4.3 Coeficiente e Fatores de Escala

O coeficiente de escala *S* é calculado a partir de uma constante *B*, que deve ser ajustada a partir de dados históricos e de um conjunto de cinco fatores de escala F_i. O cálculo é feito da seguinte forma:

$$S = B + 0{,}001 \times \sum_{j=1}^{5} F_j$$

Nesta fórmula:

- *S* é o coeficiente de escala que se deseja calcular.
- *B* é uma constante que deve ser calibrada de acordo com valores históricos. CII sugere um valor inicial de 0,91.
- F_j são cinco fatores de escala (*scale factors*) que devem ser atribuídos para cada projeto específico, tomando-se sempre muito cuidado, pois sua influência no cálculo do esforço total do projeto é exponencial.

Os cinco fatores de escala mencionados anteriormente receberão cada um uma nota que varia de "muito baixo" até "extremamente alto". Os fatores de escala terão impacto exponencial no tempo de desenvolvimento. Se seu somatório, o coeficiente de escala *S*, for nominal (= 1,0), então estima-se que um projeto com 200 KSLOC terá um esforço duas vezes maior do que um projeto com 100 KSLOC, ou seja, o crescimento do esforço é proporcional ao tamanho do projeto. Em outras palavras, mesmo que o projeto aumente de tamanho, o custo de cada linha de código permanece inalterado.

Se o coeficiente de escala ficar acima do nominal (> 1,0), então um projeto com 200 KSLOC vai usar *mais do que o dobro* do esforço do que um projeto com 100 KSLOC, ou seja, quanto maior o projeto, mais cara se torna cada linha de código.

Por outro lado, se o coeficiente de escala ficar abaixo do nominal (< 1,0), então um aumento do tamanho do projeto vai proporcionar um ganho em escala, ou seja, quanto maior o projeto, mais barata se torna cada linha de código.

Os fatores de escala são os seguintes:

- *Precedentes* (*PREC*): Se o produto é similar a vários projetos desenvolvidos anteriormente, então PREC é alto.
- *Flexibilidade no Desenvolvimento* (*FLEX*): Se o produto deve ser desenvolvido estritamente dentro dos requisitos, é preso a definições de interfaces externas, então FLEX é baixo.
- *Arquitetura/Resolução de Riscos* (*RESL*): Se existe bom suporte para resolver riscos e para definir a arquitetura, então RESL é alto.
- *Coesão da Equipe* (*TEAM*): Se a equipe é bem formada e coesa, então TEAM é alto.
- *Maturidade de Processo* (*PMAT*): Esse fator pode estar diretamente associado com o nível de

maturidade CMMI (Seção 12.3). Quanto mais alto o nível de maturidade, maior será o PMAT. As tabelas 7.12 a 7.16 são sugestões de Boehm sobre como definir a nota de cada um dos fatores de escala considerados. Para cada fator, toma-se a primeira tabela e decide-se qual a avaliação para cada uma das características na primeira coluna. Após obter todas as avaliações (uma para cada linha), determina-se uma nota para o fator como um todo, escolhendo a coluna mais representativa (muito baixo, baixo, nominal, alto, muito alto ou extremamente alto). A escolha da coluna mais representativa não é feita de maneira formal, ou seja, a ponderação dos requisitos será feita pelo planejador de projeto de acordo com sua percepção e experiência. Mas a princípio, deve-se considerar a coluna que represente a média subjetiva das notas dadas.

Uma vez selecionada a avaliação (coluna) para o fator de escala, usa-se a segunda tabela para obter sua equivalente numérica, a qual será usada na fórmula do coeficiente de escala *S*, conforme mostrado no início desta seção.

Para PREC (o produto é similar a produtos anteriormente desenvolvidos pela mesma equipe), a Tabela 7.12, em duas partes, apresenta os padrões para atribuição do equivalente numérico.

TABELA 7.12 Forma de obtenção do equivalente numérico para PREC

Característica	**Muito baixo, Baixo**	**Nominal, Alto**	**Muito alto, Extra alto**
Compreensão organizacional dos objetivos do produto	Geral	Considerável	Total
Experiência no trabalho com sistemas de software relacionados	Moderada	Considerável	Extensiva
Desenvolvimento concorrente de novo hardware e procedimentos operacionais associados	Extensivo	Moderado	Algum
Necessidade de arquiteturas e algoritmos de processamento de dados inovadores	Considerável	Algum	Mínimo

Nota média	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Interpretação	Totalmente sem precedentes	Largamente sem precedentes	Um tanto sem precedentes	Genericamente familiar	Altamente familiar	Totalmente familiar
Fator numérico	6,20	4,96	3,72	2,48	1,24	0,00

Assim, por exemplo, imaginando que um projeto tenha a seguinte avaliação:

- Compreensão organizacional dos objetivos do produto: *considerável*.
- Experiência no trabalho com sistemas de software relacionados: *moderada*.
- Desenvolvimento concorrente de novo hardware e procedimentos operacionais associados: *moderado*.
- Necessidade de arquiteturas e algoritmos de processamento de dados inovadores: *algum*.

Neste caso, tem-se três notas na coluna "Nominal/Alto" e uma nota na coluna "Muito baixo/Baixo". Não faria sentido atribuir notas "Muito baixo" (porque apenas uma nota contra três está nessa coluna), nem "Muito alto" ou "Extra alto" (porque nenhuma nota está nessas colunas). Como são três notas na coluna "Nominal/Alto" e uma na coluna "Muito baixo/Baixo", pode-se

atribuir nota "Nominal" a PREC (o limite inferior da segunda coluna). Assim, a interpretação de PREC vai considerar que o projeto é "um tanto sem precedentes" e o fator de escala numérico será 3,72.

A Tabela 7.13 apresenta a forma de cálculo do fator de escala *FLEX*, ou seja, qual a *flexibilidade no desenvolvimento* em relação aos requisitos.

TABELA 7.13 Forma de obtenção do equivalente numérico para FLEX

Característica	**Muito baixo, Baixo**	**Nominal, Alto**	**Muito alto, Extra alto**
Necessidade de conformação do software a requisitos preestabelecidos	Total	Considerável	Básica
Necessidade de conformação do software a especificações de interfaces com sistemas externos	Total	Considerável	Básica
Combinação das inflexibilidades acima com prêmio por término antecipado do projeto	Alto	Médio	Baixo

Nota média	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Interpretação	Rigoroso	Relaxamento ocasional	Algum relaxamento	Conformidade geral	Alguma conformidade	Metas gerais
Fator numérico	5,07	4,05	3,04	2,03	1,01	0,00

Interpretando as tabelas anteriores, depreende-se que quanto maior o rigor em relação à conformidade com os requisitos, ou seja, quanto menor a flexibilidade do projeto, mais tempo ele vai levar para ser desenvolvido. Isso é natural, uma vez que as atividades de projeto incluem não apenas a escrita de código, mas todas as atividades, inclusive as inspeções de conformidade e testes exaustivos.

O fator RESL, ou seja, a existência de arquitetura ou sistema de suporte para *resolução de riscos* é calculado de acordo com os dados da Tabela 7.14.

TABELA 7.14 Forma de obtenção do equivalente numérico para RESL

Característica	**Muito baixo**	**Baixo**	**Nominal**	**Alto**	**Muito alto**	**Extra alto**
O plano de gerenciamento de risco identifica todos os itens de risco críticos e estabelece marcos para resolvê-los	Nada	Um pouco	Alguma coisa	Geralmente	Largamente	Totalmente
Cronograma, orçamento e marcos internos são	Nada	Um pouco	Alguma coisa	Geralmente	Largamente	Totalmente

Característica	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
compatíveis com o plano de gerenciamento de risco						
Percentual do cronograma de desenvolvimento devotado a estabelecer a arquitetura, uma vez definidos os objetivos gerais do produto	5	10	17	25	33	40
Percentual de arquitetos de software experientes (*top*) disponíveis para o projeto em relação ao considerado necessário	20	40	60	80	100	120
Suporte de ferramentas disponível para resolver itens de risco, desenvolver e verificar especificações arquiteturais	Nenhum	Pouco	Algum	Bom	Forte	Total
Nível de incerteza nos determinantes-chave da arquitetura: missão, interface com usuário, COTS, hardware, tecnologia, desempenho	Extremo	Significativo	Considerável	Algum	Pouco	Muito pouco
Número de itens de risco e sua importância	Mais de 10 críticos	5 a 10 críticos	2 a 4 críticos	1 crítico	Mais de 5 não críticos	Menos de 5 não críticos

Nota média	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Interpretação	Pouco (20%)	Algum (40%)	Frequente (60%)	Geralmente (75%)	Largamente (90%)	Totalmente (100%)
Fator numérico	7,07	5,65	4,24	2,83	1,41	0,00

Um risco será crítico, se sua exposição for alta, ou seja, se o produto da sua probabilidade pelo seu impacto for alto, conforme mostrado na Seção 8.4. Riscos não críticos seriam os de exposição média. Já os riscos de baixa exposição não precisam ser contabilizados na última linha da tabela de RESL.

No Processo Unificado, o porcentual do cronograma devotado a estabelecer a arquitetura pode

ser compreendido como o porcentual de duração da fase de elaboração em relação ao projeto como um todo, já que essa fase específica tem como objetivo estabilizar a arquitetura.

O fator de escala TEAM, ou seja, a *coesão da equipe de desenvolvimento*, pode ser calculado como mostrado na Tabela 7.15.

TABELA 7.15 Forma de obtenção do equivalente numérico para TEAM

Característica	**Muito baixo**	**Baixo**	**Nominal**	**Alto**	**Muito alto**	**Extra alto**
Consistência dos objetivos e cultura dos interessados	Pouca	Alguma	Básica	Considerável	Forte	Total
Habilidade e vontade dos interessados em acomodar os objetivos de outros interessados	Pouca	Alguma	Básica	Considerável	Forte	Total
Experiência dos interessados em trabalhar como uma equipe	Nenhuma	Pouca	Pouca	Básica	Considerável	Extensiva
Construção de equipes com os interessados para obter visão compartilhada e compromissos	Nenhuma	Pouca	Pouca	Básica	Considerável	Extensiva

Nota média	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Interpretação	Interações muito difíceis	Algumas interações difíceis	Interações basicamente cooperativas	Interações predominantemente cooperativas	Interações altamente cooperativas	Interações perfeitas
Fator numérico	5,48	4,38	3,29	2,19	1,10	0,00

É possível observar que os fatores de escala do CII não são apenas itens para estimação de esforço, mas também recomendações de boas práticas, ou seja, objetivos a serem buscados. Se cada uma das características listadas obtiver notas positivas, o tempo de desenvolvimento tenderá a ser muito mais baixo do que com notas mais negativas.

Finalmente, o fator PMAT, ou *maturidade do processo*, pode ser calculado a partir do nível de maturidade obtido pela empresa, de acordo com o modelo CMMI (Seção 12.3) ou SPICE (Seção 12.2). O equivalente numérico pode ser obtido como mostrado na Tabela 7.16.

TABELA 7.16 Forma de obtenção do equivalente numérico para PMAT

Característica	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Nível CMMI, SPICE ou EPML	0	1	2	3	4	5

Nota média	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Interpretação	Sem processo definido	Processo incipiente	Processo definido	Processo gerenciado	Processo padronizado gerenciado quantitativamente	Processo em otimização constante
Fator numérico	7,80	6,24	4,68	3,12	1,56	0,00

Na falta de uma avaliação por CMMI ou SPICE, pode-se aplicar um questionário de avaliação EPML (Boehm, 2000) para obter o nível correspondente. EPML significa *Estimated Process Maturity Level*. O questionário e as formas de cálculo são apresentados na Seção 7.4.7

7.4.4 MULTIPLICADORES DE ESFORÇO

Os multiplicadores de esforço M_i são usados para ajustar a estimativa de esforço para o desenvolvimento de um sistema baseando-se em características próprias do projeto e da equipe que podem onerar esse tempo. Sua quantidade varia em função de se estar calculando o esforço nas fases iniciais ou intermediárias do projeto.

Durante as fases de concepção e início da elaboração usa-se o *Early Design Model* (Seção 7.4.4.2 com 6 fatores (n = 6). Mais tarde, pode-se usar o *Post-Architecture Model* (Seção 7.4.4.1), com 16 fatores (n = 16). A Figura 7.4 resume o momento em que cada um dos modelos deve ser usado, sendo que na fase de elaboração a decisão por um modelo ou outro vai depender de quanto já se tenha avançado.

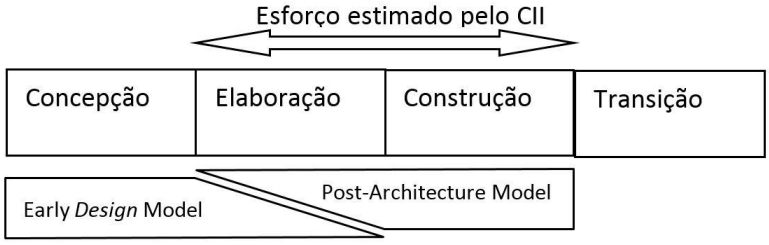

Figura 7.4 Momento de aplicação dos modelos *early design* e *post-architecture*

Nota-se que, se qualquer um dos modelos for aplicado durante a fase de elaboração ou construção, o modelo vai prever o esforço total dessas duas fases. Então, o esforço já despendido deverá ser descontado para que a previsão diga respeito ao tempo ainda restante do projeto.

7.4.4.1 Multiplicadores de Esforço do Post-Architecture Model

Os multiplicadores de esforço do *Post-Architecture Model* estão divididos nos seguintes grupos:

- Fatores do produto.

- Fatores da plataforma.
- Fatores humanos.
- Fatores de projeto.

Os *fatores do produto* avaliam características do produto que podem afetar o esforço de desenvolvimento. Esses fatores são os seguintes:

- Software com Confiabilidade Requerida (RELY).
- Tamanho da Base de Dados (DATA).
- Complexidade do Produto (CPLX).
- Desenvolvimento Visando Reusabilidade (RUSE).
- Documentação Necessária para o Ciclo de Desenvolvimento (DOCU).

O multiplicador de esforço RELY (*Software com Confiabilidade Requerida*) avalia o tipo de consequências caso o software tenha alguma falha. Para aplicar a tabela, deve-se encontrar o descritor (primeira linha) que melhor descreve o fator de confiabilidade requerida. A partir dele encontram-se a avaliação e seu equivalente numérico na coluna correspondente (Tabela 7.17). Por exemplo, quando o efeito de uma falha do software é apenas inconveniente, então RELY é *muito baixo* e o equivalente numérico é 0,82.

TABELA 7.17 Forma de obtenção do equivalente numérico para RELY

Descritor	Pequena inconveniência	Perdas pequenas, facilmente recuperáveis	Perdas moderadas, facilmente recuperáveis	Alta perda financeira	Risco à vida humana	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito Alto	Extra alto
Equivalente numérico	0,82	0,92	1,00	1,10	1,26	n/a

No caso de RELY, a avaliação "Extra alto" é inexistente (n/a). Isso também acontecerá com outros multiplicadores, como será visto mais adiante.

O multiplicador de esforço DATA (*Tamanho da Base de Dados*) avalia o tamanho relativo da base de dados usada para testes do programa (não a base de dados final). A razão *D*/*P* é o número de *Kbytes* na base de dados de teste (*D*) dividido pelo número de milhares de linhas (*P*) estimado do programa (em KSLOC). A Tabela 7.18 apresenta os parâmetros de cálculo para DATA.

TABELA 7.18 Forma de obtenção do equivalente numérico para DATA

Descritor		D/P < 10	10 ≤ D/P ≤ 100	100 ≤ D/P ≤ 1.000	D/P ≥ 1.000	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito Alto	Extra alto
Equivalente numérico	n/a	0,90	1,00	1,14	1,28	n/a

Assim, por exemplo, se o projetista estima que a base de dados para testes do sistema deverá ter

500 registros cada um com 2 K bytes em média, *D* será igual a 1.000 K bytes. Se o número de linhas de código *P* for, digamos, 12 KSLOC, então *D*/*P* = 1.000/12 = 83,333..., e assim, o valor de DATA para esse projeto será "Nominal". Como aqui se trata de ordens de grandeza, a estimação de tamanho da base de dados e mesmo das linhas de código não precisa ser muito precisa.

O multiplicador de esforço CPLX (*Complexidade do Produto*) avalia a complexidade em cinco áreas: operações de controle (estruturas de controle, recursão, concorrência e distribuição), operações computacionais (cálculos matemáticos), operações dependentes de dispositivos (entrada e saída de dados), operações de gerenciamento de dados (desde simples dados em memória até bancos de dados distribuídos) e operações de gerenciamento de interface (simples entrada de texto num extremo e realidade virtual no outro). Então, ao contrário dos multiplicadores anteriores, que são obtidos a partir de um único descritor, CPLX terá cinco descritores (um para cada área). A complexidade do produto é dada pela média subjetivamente ponderada dessas cinco áreas, conforme mostrado na Tabela 7.19.

TABELA 7.19 Forma de obtenção do equivalente numérico para CPLX

Operações de controle	Código sequencial com poucas estruturas não aninhadas. Composição simples de módulos via chamada de procedimentos ou *script*s.	Aninhamento simples de estruturas de controle. Basicamente predicados simples.	Basicamente aninhamento simples. Algum controle intermódulos. Tabelas de decisão. Chamadas ou passagem de mensagens, incluindo processamento distribuído suportado por *middleware*.	Estruturas altamente aninhadas com vários predicados compostos. Controle de fila e pilha. Processamento distribuído homogêneo. Controle de tempo real simples em processador único.	Código reentrante e recursivo. Gerenciamento de interrupção com prioridade fixa. Sincronização de tarefas. Chamadas complexas. Processamento distribuído heterogêneo. Controle de tempo real complexo em processador único.	Escalonamento de múltiplos recursos com mudança dinâmica de prioridades. Controle em nível de microcódigo. Controle complexo de tempo real distribuído.
Operações computacionais	Avaliação de expressões simples como A = B + C*(D-E).	Avaliação de expressões de nível moderado como D = SQRT (B**2-4.*A*C).	Uso de rotinas matemáticas e estatísticas-padrão. Operações básicas sobre matrizes e vetores.	Análise numérica básica: interpolação multivariada e equações diferenciais ordinárias. Arredondamento e truncamento básicos.	Análise numérica complexa, mas estruturada: equações de matrizes, equações diferenciais parciais. Paralelização simples.	Análise numérica complexa e não estruturada: análise de ruído altamente precisa, dados estocásticos. Paralelização complexa.
Operações dependentes de dispositivo	Comandos simples de leitura e escrita com formatação simples.	Sem necessidade de conhecimento de características particulares de processador ou dispositivo de E/S. E/S feita por *Get* e *Put*.	Processamento de E/S inclui seleção de dispositivo, checagem de *status* e processamento de erros.	Operações de E/S em nível físico (traduções de endereços de armazenamento físicos; buscas e leituras etc.). *Overlap* de E/S otimizado.	Rotinas para diagnóstico de interrupção. Gerenciamento de linha de comunicação. Sistemas embarcados com consideração intensiva de *performance*.	Codificação de dispositivos dependentes de tempo. Operações microprogramadas. Sistemas embutidos com *performance* crítica.
Operações de	*Arrays* simples	Arquivos	Entrada de	Gatilhos simples	Coordenação de	Estruturas

gerenciamento de dados	em memória. Simples consultas e atualizações em COTS ou banco de dados.	simples sem edição nem *buffers*. Consultas e atualizações em bancos de dados ou COTS moderadamente complexas.	múltiplos arquivos e saída em arquivo único. Mudanças estruturais simples. Edição simples. Consultas e atualizações complexas em COTS ou banco de dados.	ativados pelo conteúdo de sequências de dados. Reestruturação de dados complexa.	bancos de dados distribuídos. Gatilhos complexos. Otimização.	relacionais e de objetos dinâmicas e altamente acopladas. Gerenciamento de dados em linguagem natural.
Operações de gerenciamento de interface com usuário	Formulários de entrada simples e geradores de relatórios.	Uso de construtores de interface com usuário (GUI) simples.	Simples uso de um conjunto de *widgets*.	Desenvolvimento e extensão de conjunto de *widgets*. E/S por voz. Multimídia.	Gráficos dinâmicos 2D e 3D moderadamente complexos. Multimídia.	Multimídia complexa. Realidade virtual. Interface em linguagem natural.
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	0,73	0,87	1,00	1,17	1,34	1,74

O multiplicador de esforço RUSE (*Desenvolvimento Visando Reusabilidade*) avalia o quanto o projeto é feito pensando em gerar componentes que depois possam ser reusados. O desenvolvimento baseado em SPL (Linhas de Produto de Software – Seção 3.14), por exemplo, leva a um valor alto de RUSE. A Tabela 7.20 apresenta o padrão de atribuição de notas a esse multiplicador.

TABELA 7.20 Forma de obtenção do equivalente numérico para RUSE

Descritor		Nenhum reuso	Dentro do projeto	Dentro de um programa	Dentro de uma SPL	Entre múltiplas SPLs
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	n/a	0,95	1,00	1,07	1,15	1,24

O maior esforço de desenvolvimento quando se usa linhas de produto de software é justamente decorrente da necessidade de maior esforço para identificar e estruturar uma família de produtos, o que é mais difícil do que quando se desenvolve um produto isolado. A vantagem das linhas de produto acaba aparecendo justamente quando um número significativo de produtos é desenvolvido e a reusabilidade reduz o esforço necessário para produzir cada produto individual derivado da linha.

O multiplicador DOCU (*Documentação Necessária para o Ciclo de Desenvolvimento*) mede o quanto a documentação necessária para o desenvolvimento realmente é produzida. Se não há compromisso com a documentação, a DOCU é baixo; se há documentação em excesso, além das necessidades reais, então o índice é alto. A Tabela 7.21 mostra como chegar aos valores numéricos desse multiplicador.

TABELA 7.21 Forma de obtenção do equivalente numérico para DOCU

Descritor	Muitas necessidades de ciclo de vida não cobertas	Algumas necessidades de ciclo de vida não cobertas	Exatamente dimensionada para as necessidades do ciclo de vida	Excessiva para as necessidades do ciclo de vida	Muito excessiva para as necessidades do ciclo de vida	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	0,81	0,91	1,00	1,11	1,23	n/a

O fato de o processo de desenvolvimento deixar muitas necessidades de documentação não cobertas faz com que o desenvolvimento seja mais rápido, pois documentar leva tempo. Porém, isso usualmente não é uma boa prática, pois poderá gerar problemas mais adiante, especialmente na fase de operação do sistema, quando ele precisar sofrer manutenção.

Entretanto, documentação excessivamente burocrática exigirá muito esforço sem necessariamente produzir algum ganho depois.

Os multiplicadores de esforço referentes à *plataforma* se referem à complexidade da plataforma-alvo de implementação (hardware e software básico). Esses fatores são os seguintes:

- Restrição de Tempo de Execução (TIME).
- Restrição de Memória Principal (STOR).
- Volatilidade da Plataforma (PVOL).

O multiplicador TIME (*Restrição de Tempo de Execução*) avalia a porcentagem esperada de uso dos processadores disponíveis pela aplicação (Tabela 7.22).

TABELA 7.22 Forma de obtenção do equivalente numérico para TIME

Descritor			Menos de 50% de uso do tempo de execução disponível	70% de uso do tempo de execução disponível	85% de uso do tempo de execução disponível	95% de uso do tempo de execução disponível
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	n/a	n/a	1,00	1,11	1,29	1,63

O multiplicador STOR (*Restrição de Memória Principal*) avalia a porcentagem esperada de uso da memória principal pela aplicação. A Tabela 7.23 apresenta os equivalentes numéricos para esse multiplicador.

TABELA 7.23 Forma de obtenção do equivalente numérico para STOR

Descritor			Menos de 50% de uso da memória principal	70% de uso da memória principal	85% de uso da memória principal	95% de uso da memória principal
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	n/a	n/a	1,00	1,05	1,17	1,46

O uso de técnicas como *swap* nos modernos sistemas operacionais pode fazer que a preocupação com o uso da memória principal seja sempre nominal no caso de sistemas de informação, porque a memória sempre pode ser aumentada virtualmente. Porém, em aplicações embarcadas, esse indicador pode ser avaliado de forma mais crítica, justamente pela falta desse tipo de mecanismo.

O multiplicador PVOL (*Volatilidade da Plataforma*) avalia a plataforma de desenvolvimento, a qual inclui hardware e software básicos, sobre os quais a aplicação é construída. O fator é avaliado como "Baixo" quando ocorrem mudanças de plataforma em períodos superiores a um ano e como "Alto" quando as mudanças ocorrem em média a cada duas semanas. A Tabela 7.24 mostra como obter os valores numéricos.

TABELA 7.24 Forma de obtenção do equivalente numérico para PVOL

Descritor		Mudanças grandes a cada 12 meses, pequenas a cada mês	Grandes: 6 meses; pequenas: 2 semanas	Grandes: 2 meses; pequenas: 1 semana	Grandes: 2 semanas; pequenas: 2 dias	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	n/a	0,87	1,00	1,15	1,30	n/a

Os *fatores humanos* considerados multiplicadores de esforço são os seguintes:

- Capacidade dos Analistas (ACAP).
- Capacidade dos Programadores (PCAP).
- Continuidade de Pessoal (PCON).
- Experiência em Aplicações Semelhantes (APEX).
- Experiência na Plataforma (PLEX).
- Experiência na Linguagem e Ferramentas (LTEX).

O multiplicador ACAP (*Capacidade dos Analistas*) avalia a capacidade dos analistas de analisar e modelar aplicações, eficiência e eficácia, e as habilidades de cooperar e comunicar. Quanto maior a capacidade, menor o valor de ACAP. A Tabela 7.25 mostra como obter o valor numérico para a capacidade dos analistas, avaliada em termos de percentis. Por exemplo, se os analistas estão no percentil 15% mais baixo, então o multiplicador é avaliado como "Muito baixo".

TABELA 7.25 Forma de obtenção do equivalente numérico para ACAP

Descritor	Percentil 15	35	55	75	90	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,42	1,19	1,00	0,85	0,71	n/a

O multiplicador PCAP (*Capacidade dos Programadores*) avalia os programadores de forma semelhante à ACAP (Tabela 7.26).

TABELA 7.26 Forma de obtenção do equivalente numérico para PCAP

Descritor	Percentil 15	35	55	75	90	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,34	1,15	1,00	0,88	0,76	n/a

O multiplicador PCON (*Continuidade de Pessoal*) avalia a porcentagem de trocas de desenvolvedores no período de um ano. Quanto menos trocas, menor o valor de PCON. A Tabela 7.27 mostra como obter o valor numérico de PCON a partir da porcentagem de troca de desenvolvedores no período de um ano.

TABELA 7.27 Forma de obtenção do equivalente numérico para PCON

Descritor	48%/ano	24%/ano	12%/ano	6%/ano	3%/ano	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,29	1,12	1,00	0,90	0,81	n/a

O multiplicador APEX (*Experiência em Aplicações Semelhantes*) avalia o tempo médio (em anos) de experiência da equipe em aplicações semelhantes à que vai ser desenvolvida. Quanto maior o tempo, menor o valor de APEX. A Tabela 7.28 mostra como se obter os valores para APEX.

TABELA 7.28 Forma de obtenção do equivalente numérico para APEX

Descritor	Menos de 2 meses	6 meses	1 ano	3 anos	6 anos	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,22	1,10	1,00	0,88	0,81	n/a

O multiplicador PLEX (*Experiência na Plataforma*) avalia a experiência da equipe na plataforma de desenvolvimento, incluindo bibliotecas, hardware, sistema operacional, banco de dados, *middleware* e outros itens relacionados. A Tabela 7.29 apresenta a forma de se obter os equivalentes numéricos para PLEX.

TABELA 7.29 Forma de obtenção do equivalente numérico para PLEX

Descritor	Menos de 2 meses	6 meses	1 ano	3 anos	6 anos	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,19	1,09	1,00	0,91	0,85	n/a

O multiplicador LTEX (*Experiência na Linguagem e Ferramentas*) avalia o tempo médio de experiência da equipe nas ferramentas CASE e linguagens usadas para o desenvolvimento (Tabela 7.30).

TABELA 7.30 Forma de obtenção do equivalente numérico para LTEX

Descritor	Menos de 2 meses	6 meses	1 ano	3 anos	6 anos	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,20	1,09	1,00	0,91	0,84	n/a

Os *fatores de projeto* avaliam a influência do uso de ferramentas modernas de desenvolvimento, ambiente de trabalho e aperto do cronograma. Esses fatores são os seguintes:

- Uso de Ferramentas de Software (TOOL).
- Equipe de Desenvolvimento Distribuída (SITE).
- Cronograma de Desenvolvimento Requerido (SCED).

O multiplicador TOOL *(Uso de Ferramentas de Software*) avalia a qualidade do suporte computacional ao ambiente de desenvolvimento. O uso de simples compiladores implica um índice ruim (alto) para esse fator, enquanto o uso de ferramentas CASE e de gerenciamento de projeto que integram todas as atividades de desenvolvimento implicam uma boa avaliação (índice baixo). A Tabela 7.31 mostra como obter os valores numéricos para TOOL.

TABELA 7.31 Forma de obtenção do equivalente numérico para TOOL

Descritor	Editar, codificar, debugar.	CASE simples. Pouca integração.	Ferramentas básicas de ciclo de vida moderadamente integradas.	Ferramentas de ciclo de vida fortes e maduras, moderadamente integradas.	Ferramentas de ciclo de vida fortes, maduras e bem integradas com processos, métodos e reuso.	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,17	1,09	1,00	0,90	0,78	n/a

O multiplicador SITE (*Equipe de Desenvolvimento Distribuída*) avalia a influência da distribuição da equipe de desenvolvimento. Equipes distribuídas internacionalmente apontam para um aumento

de carga em relação a esse fator, enquanto uma equipe que trabalha toda na mesma sala implica uma carga menor. Ao contrário da maioria dos multiplicadores, SITE tem dois descritores. Deve ser feita a média subjetiva entre os dois para determinar a nota do multiplicador. A Tabela 7.32 mostra como obter o valor numérico para SITE.

TABELA 7.32 Forma de obtenção do equivalente numérico para SITE

Descritor de co-locação	Internacional	Multicidade e multiempresa	Multicidade ou multiempresa	Mesma cidade ou área metropolitana	Mesmo edifício ou complexo	Totalmente co-locada
Descritor de comunicação	Alguns telefones, correio	Telefones individuais, fax	*E-mail*	Comunicação eletrônica de banda larga	Videoconferência	Multimídia interativa
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,22	1,09	1,00	0,93	0,86	0,80

Assim, uma equipe internacional (muito baixo) que utilize videoconferência (muito alto), por exemplo, pode ser avaliada na média desses dois descritores: nominal.

O multiplicador SCED (*Cronograma de Desenvolvimento Requerido*) reflete o grau requerido de aceleração forçada a um cronograma nominal ideal ou seu relaxamento (Tabela 7.33). Um cronograma forçadamente mais rápido do que o usual implica um fator com valor mais alto, ou seja, maior esforço de desenvolvimento.

TABELA 7.33 Forma de obtenção do equivalente numérico para SCED

Descritor	75% do tempo nominal	85%	100%	130%	160%	
Avaliação	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,43	1,14	1,00	1,00	1,00	n/a

7.4.4.2 Multiplicadores de Esforço do Early Design Model

Os multiplicadores de esforço do *Early Design Model* são o resultado de combinações dos fatores do *Post-Architecture Model*, além de outras informações. Parte-se do princípio de que, quando esse modelo é aplicado, ainda não se tem muita informação sobre o real ambiente de desenvolvimento e as características do projeto. Dessa forma, as avaliações dos 17 multiplicadores de esforço do *Post-Architecture Model* são meras estimativas. Assim, o *Early Design Model* cria seus multiplicadores de esforço a partir de uma combinação dessas estimativas com outras informações mais passíveis de serem conhecidas nessa fase de um projeto.

Os sete fatores multiplicadores de esforço do *Early Design Model* são os seguintes:

- Capacidade de Pessoal (PERS).
- Confiabilidade e Complexidade do Produto (RCPX).
- Desenvolvimento para Reuso (RUSE).

- Dificuldade com a Plataforma (PDIF).
- Experiência do Pessoal (PREX).
- Instalações (FCIL).
- Cronograma de Desenvolvimento Requerido (SCED).

O multiplicador PERS *(Capacidade de Pessoal*) é obtido a partir da média subjetiva de três descritores. O primeiro deles é a soma dos multiplicadores ACAP, PCAP e PCON, definidos para o *Post-Achitecture Model*, mas considerando a seguinte equivalência numérica:

- Muito baixo = 1
- Baixo = 2
- Nominal = 3
- Alto = 4
- Muito alto = 5
- Extra alto = 6

Assim, o valor da soma dos três multiplicadores, que nesse caso variam de "Muito baixo" (1) a "Muito alto" (5) dará um resultado entre 3 e 15. O segundo descritor é o percentil combinado de ACAP e PCAP. O terceiro é o percentual anual de troca de pessoal. Observe que passa a existir a nota "Extra baixo", que não aparecia nos multiplicadores *Post-Architecture*. A Tabela 7.34 mostra como obter os valores para PERS.

TABELA 7.34 Forma de obtenção do equivalente numérico para PERS

Soma de ACAP, PCAP e PCON	3 a 4	5 a 6	7 a 8	9	10 a 11	12 a 13	14 a 15
Média dos percentis ACAP e PCAP	20%	35%	45%	55%	65%	75%	85%
Taxa de troca de pessoal anual	45%	30%	20%	12%	9%	6%	4%
Avaliação	Extra baixo	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	2,12	1,62	1,26	1,00	0,83	0,63	0,50

O multiplicador RCPX (*Confiabilidade e Complexidade do Produto*) é uma combinação dos fatores RELY, DATA, CPLX e DOCU. Ele é obtido a partir da média subjetiva de quatro descritores. A Tabela 7.35 mostra como obter os equivalentes numéricos para esse multiplicador.

TABELA 7.35 Forma de obtenção do equivalente numérico para RCPX

Soma de RELY, DATA, CPLX e DOCU	5 a 6	7 a 8	9 a 11	12	13 a 15	16 a 18	19 a 21
Ênfase em confiabilidade e documentação	Muito pouca	Pouca	Alguma	Básica	Forte	Muito forte	Extrema
Complexidade do produto	Muito simples	Simples	Alguma	Moderada	Complexa	Muito complexa	Extremamente complexa

Tamanho do banco de dados	Pequeno	Pequeno	Pequeno	Moderado	Grande	Muito grande	Muito grande
Avaliação	Extra baixo	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	0,49	0,60	0,83	1,00	1,33	1,91	2,72

O multiplicador RUSE (*Desenvolvimento para Reuso*) nessa fase é exatamente o mesmo que foi calculado para *Post-Architecture*.

O multiplicador PDIF (*Dificuldade com a Plataforma*) é uma combinação de TIME, STOR e PVOL e mais dois descritores. A Tabela 7.36 mostra como obter os valores para PDIF.

TABELA 7.36 Forma de obtenção do equivalente numérico para PDIF

Soma de TIME, STOR e PVOL			8	9	10 a 12	13 a 15	16 a 17
Restrição de tempo e memória combinadas de TIME e STOR			Menos de 50%	Menos de 50%	65%	80%	90%
Volatilidade da plataforma			Muito estável	Estável	Um tanto volátil	Volátil	Altamente volátil
Avaliação	Extra baixo	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	n/a	n/a	0,87	1,00	1,29	1,81	2,61

O multiplicador PREX *(Experiência do Pessoal*) é uma combinação de APEX, LTEX e PLEX com o descritor que se refere à experiência média da equipe. A Tabela 7.37 mostra como obter os valores para PREX.

TABELA 7.37 Forma de obtenção do equivalente numérico para PREX

Soma de APEX, PLEX e LTEX	3 a 4	5 a 6	7 a 8	9	10 a 11	12 a 13	14 a 15
Experiência média nas aplicações, plataforma, linguagem e ferramentas	Menos de 3 meses	5 meses	9 meses	1 ano	2 anos	4 anos	6 anos
Avaliação	Extra baixo	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,59	1,33	1,22	1,00	0,87	0,74	0,62

O multiplicador FCIL (*Instalações*) é uma combinação de TOOL e SITE, além de mais dois descritores. A Tabela 7.38 mostra como obter os valores para FCIL.

TABELA 7.38 Forma de obtenção do equivalente numérico para FCIL

Soma de TOOL e SITE	2	3	4 a 5	6	7 a 8	9 a 10	11
Suporte por ferramentas	Mínimo	Algum	Coleção simples de ferramentas CASE	Ferramentas básicas de ciclo de vida	Bom, moderadamente integrado	Forte, moderadamente integrado	Forte, bem integrado
Condições *multisite*	Suporte fraco em ambiente *multisite* complexo	Algum suporte para desenvolvimento *multisite* complexo	Algum suporte para desenvolvimento *multisite* moderadamente complexo	Suporte básico para ambiente *multisite* moderadamente complexo	Suporte forte para ambiente *multisite* moderadamente complexo	Suporte forte para ambiente *multisite* simples	Suporte muito forte para equipe co-locada ou ambiente multisite simples
Avaliação	Extra baixo	Muito baixo	Baixo	Nominal	Alto	Muito alto	Extra alto
Equivalente numérico	1,43	1,30	1,10	1,00	0,87	0,73	0,62

O multiplicador SCED (*Cronograma de Desenvolvimento Requerido*) é o mesmo SCED calculado para *Post-Architecture*.

7.4.5 APLICANDO CII PARA AS FASES DO UP

Como mostrado na Tabela 7.39, CII estima o esforço *E* a ser despendido nas fases de elaboração e construção. Considera-se, assim, que 100% do valor *E* estará contido nessas fases. O esforço despendido nas fases de concepção de transição será *somado* a esse valor, passando de 100%. O mesmo raciocínio aplica-se ao tempo linear *T*. As colunas "Intervalo", na Tabela 7.39, indicam os limites dentro dos quais se espera que o esforço e o tempo linear possam variar.

TABELA 7.39 Aplicação de esforço e tempo linear às fases do UP

Fase	**Esforço nominal**	**Intervalo *E***	**Tempo linear nominal**	**Intervalo *T***
Concepção	0,06*E*	0,02 a 0,15	0,125*T*	0,02 a 0,30
Elaboração	0,24*E*	0,20 a 0,28	0,375*T*	0,33 a 0,42

Fase	Esforço nominal	Intervalo *E*	Tempo linear nominal	Intervalo *T*
Construção	0,76*E*	0,72 a 0,80	0,625*T*	0,58 a 0,67
Transição	0,12*E*	0,00 a 0,20	0,125*T*	0,00 a 0,20
Totais	**1,18*E***		**1,25*T***	

Essa tabela é compatível com a Figura 6.2, em que os valores de esforço e tempo linear são convertidos em porcentagens do esforço total das quatro fases. Assim, por exemplo, um projeto com $E = 56$ desenvolvedores-mês e $T = 11{,}5$ meses terá, em média, os valores de esforço (em desenvolvedor-mês) e duração por fase definidos como na Tabela 7.40.

Tabela 7.40 Exemplo de cálculo de tempo e esforço para as fases do UP de um projeto com $E = 56$ e $T = 11{,}5$

Fase	Esforço	Intervalo *E*	Tempo	Intervalo *T*
Concepção	3,4	1,12 a 8,4	1,4	0,23 a 3,45
Elaboração	13,4	11,2 a 15,68	4,3	3,795 a 4,83
Construção	42,6	40,32 a 44,8	7,2	6,67 a 7,705
Transição	6,7	0,0 a 11,2	1,4	0,0 a 2,3
Totais	**66,1**		**14,3**	

Como se observa na Tabela 7.40, os tempos e esforços por fase podem variar dentro de faixas. O esforço e o tempo das fases de elaboração e construção devem variar inversamente, porque ambos, somados, devem resultar sempre no valor *E* e *T*, respectivamente. Mas as fases de concepção e transição podem variar livremente dentro dos intervalos definidos, pois eles não entram no cálculo de *E* e *T*. Seguem alguns exemplos de fatores que podem aumentar ou diminuir a duração dessas fases:

- Requisitos bem definidos logo no início do projeto (por exemplo, criar um substituto para um sistema que já existe) fazem que a fase de concepção seja bem menor (em tempo e esforço) do que seu valor nominal.
- Se o sistema, depois de pronto, vai mudar a maneira como as pessoas trabalham, então a fase de transição deverá ser bem maior.
- Se existem importantes riscos técnicos, a fase de concepção será maior.
- Se a comunidade de usuários é grande e heterogênea, a fase de concepção deverá ser maior.
- Se houver necessidade de integração com hardware e sistemas legados existentes, então a fase de transição será maior.

Em resumo, mais incerteza em relação aos requisitos implica uma fase de concepção maior do que o valor nominal. E implantações complexas do sistema implicam uma fase de transição maior.

O manual de CII (Boehm B., 2000) apresenta ainda outras informações, como, por exemplo, a forma de estimar o esforço relativo de cada disciplina e atividade do UP nas diferentes fases, o que é resumido na Tabela 7.41.

TABELA 7.41 Resumo do esforço relativo às disciplinas UP nas diferentes fases

Disciplina	Concepção (%)	Elaboração (%)	Construção (%)	Transição (%)
Gerenciamento	14	12	10	14
Ambiente/Configuração	10	8	5	5
Requisitos	38	18	8	4
Design	19	36	16	4
Implementação	8	13	34	19
Avaliação/Teste	8	10	24	24
Implantação	3	3	3	30
Total	**100**	**100**	**100**	**100**

7.4.6 CALIBRAGEM DO MODELO

Os valores das constantes de CII foram definidos a partir de um conjunto de projetos-base. Para obter melhores resultados em uma organização específica é necessário que essas constantes sejam calibradas para os parâmetros específicos da empresa. Além disso, com o passar do tempo, esses valores também poderão mudar, exigindo novas calibragens.

A constante *A*, por exemplo, na equação geral de estimação de esforço, tem seu valor definido em 2,94:

$$E = A \times KSLOC^S \times \prod_{i=1}^{n} M_i$$

Para se obter um valor mais adequado ao ambiente local de trabalho, recomenda-se ter realizado pelo menos cinco projetos para os quais haja a estimativa e o valor real de esforço realizado. A Tabela 7.42, adaptada de Boehm (2000), mostra os dados obtidos para oito projetos.

TABELA 7.42 Exemplo de calibragem para a constante *A* (Fonte: Boehm, 2000)

$real$	$KSLOC^S \times \sum_{i=1}^{n} M_i$	$\ln(real)$	$\ln(KSLOC^S \times \prod_{i=1}^{n} M_i)$	$\ln(real) - \ln(KSLOC^S \times \prod_{i=1}^{n} M_i)$
1854,6	686,7	7,53	6,53	0,99
258,5	94,3	5,55	4,55	1,01
201,0	77,7	5,30	4,35	0,95
58,9	20,3	4,08	3,01	1,07
9661,0	3338,8	9,18	8,11	1,06
7021,3	2753,5	8,86	7,92	0,94

$real$	$KSLOC^S \times \sum_{i=1}^{n} M_i$	$\ln(real)$	$\ln(KSLOC^S \times \prod_{i=1}^{n} M_i)$	$\ln(real) - \ln(KSLOC^S \times \prod_{i=1}^{n} M_i)$
91,7	38,9	4,52	3,66	0,86
689,7	301,1	6,54	5,71	0,83
				X = 0,96
				A = 2,62

A primeira coluna (*Real*) mostra o esforço real de cada projeto em desenvolvedor-mês (fases de elaboração e construção). A segunda coluna apresenta o cálculo da estimativa não ajustada, ou seja, a fórmula do esforço total sem a constante *A*:

$$KSLOC^S \times \prod_{i=1}^{n} M_i$$

Esses são os valores básicos para o cálculo.

As duas colunas seguintes apresentam o logaritmo natural (*ln*) do valor real do esforço e da estimativa não ajustada. Por fim, a última coluna apresenta a diferença entre essas duas colunas.

No final da tabela, o valor *X* corresponde à média das diferenças (última coluna). A constante *A* é calculada como o antilogaritmo dessa média, ou seja, $A = e^x$. Esse exemplo mostra que nesse ambiente local a constante *A* deveria ser 2,62, e não 2,94.

Boehm (2000) também mostra como calibrar a distribuição das estimativas por atividade e fase, para o ambiente local, o que pode ser particularmente interessante quando essas estimativas são efetivamente usadas para calcular o esforço dentro das iterações.

7.4.7 QUESTIONÁRIO EPML

O questionário EPML (*Estimated Process Matury Level*) é uma interessante ferramenta não só para avaliar o fator PMAT, mas também como autoavaliação da empresa em relação às boas práticas de processo. Principalmente por essa segunda razão, ele é reproduzido nesta seção.

O questionário subdivide-se em 18 áreas-chave, correspondentes às áreas de avaliação de processo do CMMI. Cada pergunta deve ser respondida com uma das opções conforme a Tabela 7.43.

Tabela 7.43 Opções de resposta às perguntas do questionário EPML

Resposta	**Explicação**	**Frequência**	**Equivalente numérico**
Quase sempre	Quando os objetivos são consistentemente obtidos e bem estabelecidos em procedimentos operacionais padrão	Mais de 90% das vezes	1,0
Frequentemente	Quando os objetivos são obtidos com relativa frequência, mas algumas vezes são omitidos por conta de circunstâncias difíceis	Entre 60 e 90% das vezes	0,75
Metade do tempo	Quando os objetivos são obtidos em cerca de metade das vezes	Entre 40 e 60% das vezes	0,50
Ocasionalmente	Quando os objetivos são obtidos algumas vezes, mas com pouca frequência	Entre 10 e 40% das vezes	0,25
Raramente	Quando os objetivos raramente ou nunca são obtidos	Menos de 10% das	0,01

		vezes	
Não se aplica	Quando se tem o conhecimento necessário sobre a organização, o projeto e a área-chave, mas entende-se que área-chave não se aplica às circunstâncias		
Não sabe	Quando não se sabe o que responder à área-chave		

Deve ser, então, dada uma resposta para cada uma das 18 áreas-chave, conforme a Tabela 7.44.

Tabela 7.44 Perguntas do questionário EPML

Área	**Perguntas**
Gerenciamento de requisitos	Os requisitos do software são controlados a ponto de se estabelecer uma *baseline* para uso da gerência de engenharia de software? Os planos, produtos e atividades relacionados ao software são mantidos consistentes com os requisitos do software?
Planejamento de projeto de software	Estimativas de esforço são documentadas para uso em planejamento e rastreamento de projeto de software? As atividades são planejadas e documentadas? Grupos afetados e indivíduos concordam formalmente com seus entendimentos (*commitments*) relacionados ao projeto de software?
Rastreamento e supervisão de projeto de software	Resultados de desempenho reais são rastreados em relação aos planos de software? Ações corretivas são tomadas e gerenciadas até o final, quando os resultados e desempenho reais se desviam significativamente dos planos de software? Mudanças nos entendimentos do software são acordadas formalmente pelos grupos ou indivíduos afetados?
Gerenciamento de subcontratos de software	O contratador seleciona subcontratados qualificados? O contratador e o subcontratado concordam formalmente com seus entendimentos um com outro? O contratador e o subcontratado mantêm comunicações constantes? O contratador rastreia os resultados e desempenho reais do subcontratado em relação aos seus empreendimentos?
Garantia de qualidade de software	As atividades de garantia de qualidade de software são planejadas? A aderência de produtos e atividades de software aos padrões, procedimentos e requisitos aplicáveis é verificada objetivamente? Grupos e indivíduos afetados são informados das atividades e resultados de garantia de qualidade de software? Assuntos de não conformação que não podem ser resolvidos dentro do projeto de software são levados à gerência superior?
Gerenciamento de configuração de software	As atividades de gerenciamento de configuração de software são planejadas? Os produtos de trabalho selecionados são identificados, controlados e disponibilizados? Mudanças nos produtos de trabalho identificadas são controladas? Os grupos e indivíduos afetados são informados do *status* e do conteúdo das *baselines* de software?
Foco do processo organizacional	Atividades de desenvolvimento e melhoria de processo de software são coordenadas entre as diferentes partes da organização? Os pontos fortes e fracos do processo de software usado são identificados em relação a um processo-padrão? As atividades de desenvolvimento e melhoria de processo em nível organizacional são planejadas?
Definição de processo organizacional	É desenvolvido e mantido um processo de software padrão para a organização? Informação relacionada ao uso do processo de software padrão pelos projetos é coletada, revisada e disponibilizada?
Programa de treinamento	As atividades de treinamento são planejadas? É fornecido treinamento para o desenvolvimento de habilidades e conhecimentos necessários para realizar a gerência e os papéis técnicos na área de software? Os indivíduos do grupo de engenharia de software e dos grupos relacionados recebem o treinamento necessário para desempenhar seus papéis?
Gerenciamento integrado de software	O processo de software definido para o projeto é uma versão especializada do processo de software padrão da organização? O projeto é planejado e gerenciado de acordo com o processo de software definido para ele?
Engenharia de produto de software	As atividades de engenharia de software são definidas, integradas e consistentemente realizadas para produzir o software? Os produtos de trabalho são consistentes uns com os outros?
Coordenação intergrupos	Os requisitos do usuário são acordados por todos os grupos afetados? Os entendimentos entre os grupos de engenharia são acordados pelos outros grupos afetados? Os grupos de engenharia identificam, rastreiam e resolvem assuntos intergrupos?

Revisões	Atividades de revisão são planejadas? Defeitos nos produtos de trabalho são identificados e removidos?
Gerenciamento quantitativo de processo	As atividades de gerenciamento quantitativo de processo são planejadas? O desempenho de processo para o projeto definido é controlado quantitativamente? A capacidade do processo de software padrão da organização é conhecida em termos quantitativos?
Gerenciamento da qualidade de software	As atividades de gerenciamento de qualidade do projeto de software são planejadas? Objetivos mensuráveis para a qualidade de produto de software e suas prioridades são definidos? O progresso real rumo a obter as metas de qualidade para os produtos de software é quantificado e gerenciado?
Prevenção de defeitos	Atividades de prevenção de defeitos são planejadas? Causas comuns de efeitos são detectadas e identificadas? Causas comuns de defeitos são priorizadas e sistematicamente eliminadas?
Gerenciamento de mudança tecnológica	A incorporação de mudanças na tecnologia é planejada? Novas tecnologias são avaliadas para determinar seu efeito na qualidade e produtividade? Novas tecnologias apropriadas são transferidas e praticadas normalmente por toda a organização?
Gerenciamento de mudança de processo	A melhoria de processo contínua é planejada? Toda a organização participa das atividades de melhoria de processo de software? O processo de software padrão da organização e os projetos definidos são continuamente melhorados?

Para obter o valor de EPML (entre 0 e 5) que vai permitir avaliar o multiplicador PMAT é necessário transformar as respostas dadas anteriormente em um número. Inicialmente, eliminam-se áreas-chave para as quais a resposta foi "Não se aplica" ou "Não sei". Fica-se, então, com *n* respostas válidas. Mas deve-se tomar cuidado, porque quanto menor o *n*, menos confiança se poderá ter no resultado da avaliação. A princípio, as 18 áreas deveriam ser avaliadas.

As respostas dadas são convertidas em equivalentes numéricos K_i (para $1 \leq i \leq n$), conforme indicado na Tabela 7.43. Então, EPML é calculado através da seguinte fórmula:

$$EPML = 5 \times \frac{(\sum_{i=1}^{n} K_i)}{n}$$

Finalmente, o valor é arredondado para o valor inteiro mais próximo e aplicado na Tabela 7.16 para obter-se a avaliação de PMAT.

REMISSIVO DO CAPÍTULO