A Estimativa de Esforço Baseada em Casos de Uso

John Smith

Rational Software White Paper

fd0201pcTP 171, 10/99

Rational®
the software development company

Índice Analítico

O Problema

Intuitivamente, parece ser possível formar estimativas de tamanho e esforço que o desenvolvimento precisará com base nas características do modelo de caso de uso. Finalmente, o modelo de caso de uso captura os requisitos funcionais, portanto, não deve haver um equivalente de pontos de funções baseado em caso de uso? Existem várias dificuldades:

- Existem muitas variações de estilo e formalidade de especificação de caso de uso, o que torna muito difícil definir métricas—alguém pode gostar, por exemplo, de medir o comprimento de um caso de uso.
- Casos de uso devem representar a visão de um sistema por um agente externo e, portanto, um caso de uso para um sistema de software com 500.000 linhas de código (sloc) está em um ***nível*** bem diferente de um caso de uso escrito para um subsistema de 5.000 sloc (Cockburn97 discute a noção de níveis e objetivos).
- Casos de uso podem diferir em complexidade, explicitamente quando escritos e implicitamente na realização requerida.
- Um caso de uso deve descrever o comportamento de ponto de vista do agente, mas isso pode ser muito complexo, especialmente se o sistema tiver estados (como tem a maioria). Portanto, para descrever esse comportamento poderá ser necessário um modelo do sistema (antes de qualquer realização ser feita). Isso pode levar a vários níveis de decomposição funcional e detalhes, na tentativa de capturar a essência do comportamento.

Portanto, é necessária a realização de algum tipo de caso de uso para tornar a estimativa possível? Talvez, as expectativas sobre estimativa diretamente de casos de uso sejam muito altas e o desenho de paralelas entre pontos de função e uma noção de pontos de caso de uso fique mal orientado. O cálculo de contagens de pontos de função requer de qualquer maneira um modelo do sistema. A derivação de pontos de função de descrições de caso de uso precisará de uma uniformidade de nível na expressão de caso de uso e é apenas quando a realização começa a surgir que deve-se ter muita confiança em uma contagem de pontos de função. Fetcke97 descreve um mapeamento de caso de uso para pontos de função, mas, novamente, o nível do caso de uso precisa estar adequado para que o mapeamento seja válido. Outros métodos utilizam métricas baseadas em classe de uso/objeto como uma fonte, Pontos de Objeto PRICE, por exemplo, (Minkiewicz96).

Outros Trabalhos

Existe uma vasta quantidade de trabalhos sobre a descrição e a formalização de casos de uso—Hurlbut97 tem uma boa pesquisa de opinião. Há um pouco menos sobre métricas de estimativa de derivação de casos de uso. Graham95 e Graham98 contém críticas severas de casos de uso (mas não entendo por que ele acredita que suas idéias e casos de uso estão tão separados) e propôs a idéia de 'script de tarefa' como uma forma de superar os problemas com casos de uso, incluindo complexidade e comprimento variáveis. O 'script de tarefa diminuto' de Graham é a base para a coleta de uma métrica de 'ponto de tarefa'. O problema com um script de tarefa diminuto é que ele é de nível muito baixo: de acordo com Graham, o ideal é que ele fosse uma única sentença e não mais decomposto, utilizando apenas terminologia de domínio. As 'tarefas-raiz' de Graham contêm um ou mais scripts de tarefa diminutos e cada tarefa-raiz corresponde "a exatamente uma operação de sistema: na classe que inicia o plano"(Graham98). Essas tarefas-raiz se parecem muito com casos de uso de nível baixo para mim e os scripts de tarefa diminutos com etapas em tal caso de uso. O problema de nível ainda permanece.

Outro trabalho foi feito por Karner (Karner93), Major (Major98), Armour e Catherwood (Armour96) e Thomson (Thomson94). O documento de Karner apresenta um método para calcular pontos de caso de uso, mas, novamente, assume que os casos de uso são expressos de uma forma realizável pelas classes (ou seja, em um nível mais sutil de detalhes dos que os subsistemas).

Portanto, devemos evitar casos de uso para estimativa e passar a confiar nas realizações de análise e design que surgem? O problema é que isso atrasa a habilidade de fazer estimativas e não será satisfatório para um coordenador de projeto que escolheu essa tecnologia—estimativas precoces ***serão*** requeridas e outros métodos terão então de ser utilizados. É melhor para o coordenador de projeto poder obter estimativas precoces para planejar propósitos e, então, ***refiná-los*** em cada iteração, em vez de atrasar a estimativa e continuar de forma não planejada.

O que está descrito neste documento é uma estrutura na qual os casos de uso em qualquer nível podem ser utilizados para formar uma estimativa de esforço. Para apresentar as idéias, algumas estruturas canônicas estão descritas, com dimensões e tamanhos associados que têm alguma base em experiência. O documento está cheio de conjecturas audazes (ou deveria ser

atrevidas) porque não consigo ver outra maneira de continuar dada a falta de trabalhos e dados nesta área. Extrai a idéia de 'sistemas de sistemas interconectados' na formulação.

Em seguida, irei afastar-me um pouco para estabelecer algumas idéias de segundo plano que me levam para esse caminho.

Evitando Decomposição Funcional?

A idéia de decomposição funcional parece ser um maldição para muitos em desenvolvimento de software. E minha experiência pessoal em decomposição funcional levada a um extremo (três mil transformações primitivas em um fluxograma de dados muito grande, cinco ou seis níveis de profundidade, feito sem idéia de arquitetura, exceto no nível da infra-estrutura) não me fez sentir otimista sobre isso. No entanto, o problema nesse caso não foi apenas com a decomposição funcional, mas também com a idéia de não descrever um processo até o nível primitivo funcional ser atingido, ponto no qual a especificação deve ter menos de uma página de comprimento.

O resultado é muito difícil de entender—como o comportamento desejado requerido em um nível mais alto surge dessas transformações primitivas é difícil de discernir. Além disso, não fica óbvio como a estrutura funcional deve ser mapeada para uma estrutura física que atenderá ao requisito de desempenho e a outros de qualidade. Portanto, o paradoxo foi que nós decompusemos e decompusemos até chegarmos ao nível no qual poderíamos 'resolver o problema' (o nível primitivo), mas não ficou claro ou demonstrável que os primitivos que trabalham juntos realmente atendam aos objetivos em níveis mais altos. Não houve uma maneira nesse método para levar em conta requisitos não funcionais. A arquitetura, em sua totalidade, não apenas a infra-estrutura (comunicações, sistema operacional, etc.) deve ter evoluído junto com a decomposição e cada uma deve ter influenciado a outra.

E sobre a postura de Bauhaus que o 'formato segue a função'? Bem, há muitas coisas boas que fluíram de sua abordagem funcionalista para design, mas algumas ruins também, como o uso de telhados planos em qualquer lugar. Se você tiver considerado apenas a função de um telhado e subordinado o design totalmente ao telhado como sendo uma cobertura para os habitantes, então, o resultado, pelo menos em determinadas áreas, será insatisfatório. Tais telhados dificilmente são à prova d'água; eles juntarão muita neve.

Agora esses problemas podem ser resolvidos***, mas a um custo maior do que se você tivesse escolhido um design diferente***. Portanto, embora pareça banal dizer isso, o formato deve seguir os *requisitos*—todos eles, funcionais e não funcionais, e estes podem incluir estética. O problema para o arquiteto freqüentemente será que os requisitos não funcionais são, em geral, estabelecidos de forma mais simples e muita confiança é colocada na experiência do arquiteto de 'como as coisas devem ser'. Portanto, a decomposição funcional será ruim se ela conduzir exclusivamente a arquitetura—se a decomposição continuar em vários níveis para baixo e os originais funcionais forem mapeados um-a-um com 'módulos'—e definir suas interfaces.

Considerações como esta me convenceram de que não faria sentido decompor casos de uso para um nível normalizado (isso poderia ser realizado por uma colaboração de classes) ***antes do trabalho arquitetural.*** É certo que a decomposição ocorrerá se o sistema for de um certo tamanho (consulte Jacobson97), mas os critérios e processo de engenharia para decomposição são importantes—para esse fim específico, a decomposição funcional não é boa o suficiente.

Considerações do Sistema

Os engenheiros de sistema fazem análise funcional, decomposição e alocação (quando sintetizam um design)—mas a função não é única condutora para a arquitetura—equipes de engenheiros especialistas contribuirão na avaliação de designs alternativos. O IEEE Std 1220, Standard for Application and Management of the Systems Engineering Process, descreve o uso da decomposição funcional na seção *6.3, Análise Funcional* na subseção *6.3.1 Decomposição Funcional* e soluções para o produto de sistema na seção *6.5 Síntese*. As subseções são de particular interesse *6.5.1 Agrupar e Alocar Funções* e *6.5.2 Alternativas de Soluções Físicas*. Na seção *6.3.1*, é dito que a decomposição é executada para ***entender*** claramente o que o sistema deve executar e, ***em geral, um nível de decomposição é suficiente.***

Observe que a finalidade da decomposição funcional não é dar forma ao sistema (a síntese faz isso), mas entender e comunicar o que o sistema deve fazer—um modelo funcional é uma maneira válida de fazer isso. Na síntese, as subfunções são alocadas para estruturas da solução e, em seguida, a solução é avaliada—levando em conta todos os outros requisitos. A diferença entre essa abordagem e a decomposição funcional em vários níveis é que em cada nível você tenta descrever o comportamento requerido e achar uma solução para implementá-lo, antes de decidir se o comportamento no próximo nível precisa ser mais refinado e alocado para componentes de nível mais baixo.

Uma conclusão para isso é que não é necessário ter centenas de casos de uso para descrever o comportamento em qualquer nível. O número de casos de uso externos (e cenários associados) que abrangerá adequadamente o comportamento do item

descrito—sistema, subsistema, classe—pode ser muito pequeno. Devo explicar o que quero dizer com caso de uso externo. Use o exemplo de um sistema composto de subsistemas que, por sua vez, são compostos de classes. Chamei de casos de uso externos aqueles que descrevem o comportamento do sistema e de seus agentes. Os subsistemas podem ter seus próprios casos de uso—esses casos de uso são internos para o sistema, mas externos para o subsistema. O número total de casos de uso, ***externos e internos***, finalmente utilizados para construir um sistema muito grande (digamos, mais de 1.000.000 de linhas de código) poderá ser de centenas, porque os sistemas desse tamanho serão construídos como sistemas de sistemas ou, pelo menos, sistemas de subsistemas.

Premissas sobre Estrutura e Tamanho

Número de Casos de Uso

Na Rational® Software, geralmente aprendemos que o número de casos de uso deve ser pequeno (10–50) e observamos que um número grande (acima de 100) de casos de uso poderá indicar um lapso na decomposição funcional, na qual o caso de uso não está entregando nada de valor para um agente. Contudo, encontramos números grandes de casos de uso em projetos reais e nem todos são 'ruins'—eles abrangem uma mistura de níveis—por exemplo, no e-mail interno da Rational, o agente cita um exemplo da Ericsson:

> A Ericsson, modelando grandes partes de uma nova geração de central telefônica, estimou ter mais de 600 equipes-anos (no pico, 3–400 desenvolvedores), 200 casos de uso ***(utilizando mais de um nível de casos de uso, consulte "Sistemas de Sistemas Interconectados")*** (my italics)

Para um sistema de mais de 600 equipe-anos (qual é o tamanho disso? 1.500.000 linhas de código C++?). Eu suspeito que a análise do caso de uso parou em um nível acima do subsistema (ou seja, e alguém definir que um subsistema tenha 7000–10000 linhas de código), caso contrário, a contagem ainda teria sido maior.

Portanto, continuarei a noção de que um número menor de casos de uso *externos* é adequado. Para corresponder as estruturas e dimensões que propus, estou afirmando que ***10 casos de uso externos,*** cada um com ***30 cenários associados***[1] são adequados para descrever o comportamento[2]. Se em um exemplo real, o número de casos de uso exceder 10 e, em seguida, eles ficarem verdadeiramente externos a esse nível, então, o sistema que está sendo descrito é maior que a forma canônica correspondente. Tentarei fornecer algum suporte que demonstre que esses números são sensíveis posteriormente no documento.

Hierarquia Estrutural

A hierarquia estrutural proposta é:

4 — SystemOfSystems

3 — Sistema

2 — SubsystemGroup

1 — Subsistema

0 — Classe

Classe e Subsistema são definidos no UML; os agregados maiores são subsistemas (contendo subsistemas) no UML. Dei a eles nomes diferentes para facilitar a discussão. O subsystemGroup agregado tem tamanho semelhante ao CSCI, para aqueles que conhecem a terminologia de padrões militares como 2167 ou 498 (o que tornaria um subsistema um CSC e uma classe um CSU). Conforme me recordo, depois dos argumentos utilizados nos 2167 dias sobre o que a Ada construiu dever ser mapeado

[1] No UML1.3, um cenário é descrito como: "**cenário:** uma seqüência específica de ações que ilustra comportamentos. Um cenário pode ser utilizado para ilustrar uma interação ou a execução de uma instância de caso de uso". Ele é utilizado aqui no segundo sentido de ilustrar a execução de uma instância de caso de uso.

[2] Observe que esse número (de cenários) deve refletir a complexidade de um caso de uso—não é sugerido que um desenvolvedor ***deva*** produzir e anotar 30 cenários para cada caso de uso—em vez disso, 30 cenários capturam a maioria do comportamento interessante para um caso de uso, embora possa haver muitos mais caminhos através do caso de uso

para qual nível, quando a poeira baixava, o pacote Ada normalmente era mapeado para CSU. Não estou sugerindo que os sistemas devem estar rigorosamente em conformidade com essa hierarquia—haverá mistura entre os níveis—mas a hierarquia permite deduzir sobre o efeito do tamanho no esforço por caso de uso.

Haverá casos de uso em cada nível (embora provavelmente não para uma classe individual), mas não uma única massa de detalhes incríveis, em vez de casos de uso para cada componente (ou seja, subsistema, subsystemGroup, etc.) nesse nível[3]. Eu afirmei acima que deve haver 10 casos de uso para cada componente em cada nível. Se as descrições de casos de uso utilizarem em média 10 páginas, isso fornecerá um comprimento potencial para um documento de especificação de 100 páginas (mais um número similar ou menor para requisitos não funcionais). Esse é um número parecido com o de Stevens98 e está próximo ao sugerido em Royce98. Mas por que 10 casos de uso? Para chegar a esse número, eu deduzi de baixo para cima, com base no que pensava ser tamanhos razoáveis para número de classes por subsistema, tamanho de classe, tamanho da operação e assim por diante. Eles são coletados em conjunto para referência com outras premissas na seguinte tabela.

Tamanho da operação	70 slocs
Número de operações por classe	12
Número de classes por subsistema	8
Número de subsistemas por subsystemGroup	8
Número de subsystemGroups por sistema	8
Número de sistemas por systemOfSystems	8
Número de casos de uso externos (por sistema, subsistema, etc.)	10
Número de cenários por caso de uso	30
Páginas por descrição de caso de uso[4]	10

Não tenho muitos dados empíricos—existem pedacinhos e partes espalhadas em todos os textos. Lorentz94 e Henderson-Sellers96 têm alguns dados e eu tenho alguns dados de projetos na Austrália, principalmente no domínio aeroespacial militar. De qualquer forma, era importante neste estágio obter apenas a estrutura posicionada mais ou menos no local correto.

Tamanho de Componentes na Hierarquia

Devo dizer que utilizei linhas de código sabendo que algumas pessoas não gostam da medida. Elas são linhas de código do C++ (ou linguagem de nível equivalente), portanto, seria fácil o suficiente engasgar em pontos de função.

Deve haver algum relacionamento entre o número de classes em um contêiner e a riqueza do comportamento que pode ser expresso. Escolhi oito classes/subsistema[5], oito subsistemas/subsystemGroup, oito subsystemGroups/sistema e assim por diante. Por que oito?

- Está dentro de sete, mais ou menos 2.

[3] Alguns revisores ficaram assustados com a probabilidade de casos de uso em quatro níveis, mas observe que isso será apenas para um sistema de sistemas, que normalmente será muito grande. Em tais casos, eu não ficaria surpreso de ver casos de uso em quatro níveis, particularmente, se o trabalho for feito por uma contratada principal (para o sistema dos sistemas), subcontratadas (para os sistemas) e, talvez, até sub-subcontratadas para os subsistemas.

[4] Mais adiante neste documento, isso será refinado para classes de sistema diferentes.

[5] Acredito que esse tipo de contagem é representativo de análise—haverá uma expansão e recriação através de design e implementação e o número de classes aumenta em um fator de três ou mais, enquanto o tamanho da operação e o tamanho da classe diminui correspondentemente.

- Devido a 850 slocs de C++ por classe (12 operações de 70 slocs cada), o subsistema fica com um tamanho de ~7000 slocs—uma parte da funcionalidade/código que é distribuível por uma equipe pequena (digamos, equipe de 3–7) em 4–9 meses, que deve harmonizar com o comprimento de iteração de sistemas no intervalo de 300.000–1.000.000 slocs (RUP99).[6]

Portanto, qual é o número de casos de uso que expressará o comportamento (externamente) de oito classes, que são coesivas e foram co-localizadas em um subsistema? Não é simplesmente o número de casos de uso, mas também o número de cenários para cada caso de uso que determina a riqueza. Agora, não há muito no caminho de diretrizes para expansão de cenários/casos de uso—Grady Booch indica em Booch98 que: "Há um fator de expansão de casos de uso para cenários. Um sistema modestamente complexo tem algumas dúzias de casos de uso que capturam seu comportamento e cada caso de uso poderá se expandir para várias dúzias de cenários…" e Bruce Powel Douglass diz em Douglass99, "…. muitos cenários são requeridos para elaborar completamente um caso de uso—normalmente, uma dúzia para várias dúzias". Escolhi 30 cenários/casos de uso—isso está no lado inferior de 'várias dúzias", mas Rechtin (em Rechtin91) diz que engenheiros podem manipular 5–10 variáveis de interação (que para a finalidade desse argumento eu interpreto como 5–10 classes em uma colaboração) e 10–50 interações (que interpretei como cenários). Interpretado dessa maneira, vários casos de uso são várias instâncias desse espaço de variável.

Portanto, em 10 casos de uso, cada um com 30 cenários, digamos que o total de 300 cenários (que posteriormente levará a ~300 etapas de teste) é suficiente para abranger o comportamento interessante de oito classes. Há alguma outra indicação de que isso é um número razoável? Se a regra 80–20 de Pareto for aplicada, então, 20% das classes entregarão 80% da funcionalidade e, similarmente, 80% da funcionalidade serão entregues por 20% das operações em cada classe. Sejamos conservadores e digamos que precisamos de 20% de classes, etc., para atingir 75% da capacidade e construir uma distribuição de Pareto através desse ponto (Figura 1).

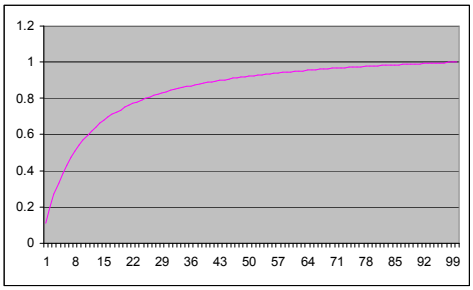


Figura 1: Uma Distribuição Semelhante a de Pareto

Se desejarmos 80% de cobertura do comportamento global e a regra de Pareto se aplicar ao número de classes, operações e cenários, então, precisaremos de 93% ($0{,}93^3$ é 0,8) de cobertura comportamental de cada um—ou seja, requerer 50% de cada; isto é, 4 classes e 5 operações (= (12 menos 2 construtores/destruidores)/2). O número de passagens diferentes dos três nós construídos para representar os padrões de execução de quatro classes com cinco operações cada uma poderá ser executado para vários milhares. Eu construí uma com até três links de cada nó, assumindo uma hierarquia, com 10 operações (operações de interface) na parte superior e formando uma árvore de três níveis. Isso fornece quase 1000 caminhos ou cenários. Portanto, 500 cenários devem fornecer 93% de cobertura. Com 300 cenários (utilizando as mesmas premissas) devemos obter cerca de 73% de cobertura. Examinando como a árvore pode ser aparada, para eliminar especificação comportamental redundante, sugiro que números ainda menores poderão ser adequados, dependendo do algoritmo escolhido.

Outra forma de abordar isso é perguntar quantos casos de teste (derivados de cenários) deve-se esperar para 7000 slocs de C++. Esses testes seriam qualquer coisa além do nível de teste da unidade e há alguma evidência de Jones91 e do projeto Boeing 777 (Pehrson96) que esse número é seguro, pelo menos, ele representa a prática. Essas fontes sugerem que entre 250–

[6] Para sistemas menores (tempos de iteração mais curtos), os subsistemas podem ser planejados para serem menores ou sempre é possível planejar uma entrega parcial para cada iteração—embora isso precise de controle cuidadoso e possa requerer a entrega de 'stubs'.

280[7] estão quase corretos. Em um nível completamente diferente, o projeto CAATS (Canadian Automated Air Traffic System) utiliza 200 testes do sistema (comunicação privada).

Tamanho do Caso de Uso

Qual tamanho deve ter um caso de uso? Ele deve ser grande o suficiente para apresentar detalhes suficientes para que o comportamento desejado possa ser realizado—isso dependerá de sua complexidade, interna e externa, que estará relacionada ao tipo de sistema. Aqui entramos no problema de quanto da ação interna de um sistema deve ser descrito. Construir um sistema a partir de uma descrição de seu comportamento requer, obviamente, que as saída estejam relacionadas às entradas. Agora se, por exemplo, o comportamento for sensível ao histórico e complexo, será muito difícil descrevê-lo sem algum modelo conceitual da parte interna do sistema e das ações que ele executa. Observe, no entanto, que isso não descreve necessariamente como o sistema deve ser construído internamente—qualquer design que satisfaça os requisitos não funcionais e que corresponda ao comportamento do modelo o fará.

A definição oferecida no UML1.3 é: "**caso de uso [classe]:** a especificação de uma seqüência de ações, incluindo variantes, que um sistema (ou outra entidade) pode executar, interagindo com agentes do sistema". Para comportamento complexo, essa definição pode ser razoavelmente obtida para incluir ações internas—a menos que isso seja adiado até a realização—que está uma etapa além do usuário final. As regras de negócios devem ser incorporadas em casos de uso para restringir o comportamento dos agentes; por exemplo, em um sistema ATM, um banco poderá ter uma regra que não mais que U$500 dólares poderá ser retirado em uma única transação, não importando o saldo da conta.

Com esse tipo de interpretação, a descrição do fluxo de eventos do caso de uso pode variar entre 2–20[8]. Sistemas algoritmicamente simples com comportamento simples não precisarão obviamente de descrições longas. Talvez, possamos dizer que sistemas de negócios simples sejam caracterizados em 2–10 páginas, com uma média de 5. Sistemas mais complexos, negócios e científicos em 6–15 páginas, com uma média de 9, e comando e controle complexos em 8–20 páginas, com uma média de 12 (essas proporções refletem o relacionamento não linear do esforço para o tipo de sistema de mesmo tamanho) embora eu não tenha dados para backup. Formas descritivas mais expressivas, máquinas de estado ou diagramas de atividade, por exemplo, podem tomar menos espaço. Ainda tendemos a enfatizar o texto, portanto, ignoraremos os outros por agora—de qualquer forma, não há dados ou muito pouco.

Os desenvolvimentos que diferem sistematicamente desses tamanhos devem aplicar um multiplicador para as horas por caso de uso derivado desses heurísticos (sugiro a inclusão de um condutor de custo do estilo COCOMO, que é o tamanho médio observado/tamanho médio sugerido para a classificação do sistema—negócio simples, mais complexo, comando e controle, etc.).

Outro aspecto do tamanho de caso de uso é a contagem de cenários; por exemplo, um caso de uso que tem somente 5 páginas de comprimento poderá ter uma estrutura complexa que permitirá muitos caminhos. Novamente, o número de cenários precisa ser estimado e a proporção disso para trinta (minha suposição inicial de um número de cenários por caso de uso) utilizado como um condutor de custo.

A conseqüência é que estamos afirmando que uma especificação baseada em caso de uso de ~100 páginas deve ser suficiente para uma especificação externa em qualquer nível determinado, além da especificação suplementar. O intervalo é de 20–200 páginas (esses limites são vagos). Observe, no entanto, que o total para um sistema (de subsystemGroups) ***no nível mais inferior*** é 3–15 páginas/ksloc (sistema de negócios simples)—12–30 páginas/ksloc (comando e controle complexos). Isso parece explicar a contradição aparente entre a Tabela 14-9 de Royce98, na qual as contagens de páginas para artefatos são muito pequenas e a observação de projetos reais, que, particularmente, em defesa produziram grandes quantidades de papel. Este documento vem de um nível de especificação que precisa ser confirmado para ser escrito—Royce está certo, as coisas importantes, como a Declaração de Visão devem ter a ordem indicada na tabela—200 páginas para sistemas grandes e complexos.

---

[7] A partir do feedback que recebi de revisores dentro da Rational, o que se sente é que isso é mais do que suficiente para a maioria dos sistemas não-críticos; que esses sistemas terão menos de 30 cenários por caso de uso. Seria interessante ter mais dados sobre isso e o relacionamento entre números de casos de teste e o número de defeitos descobertos em uso.

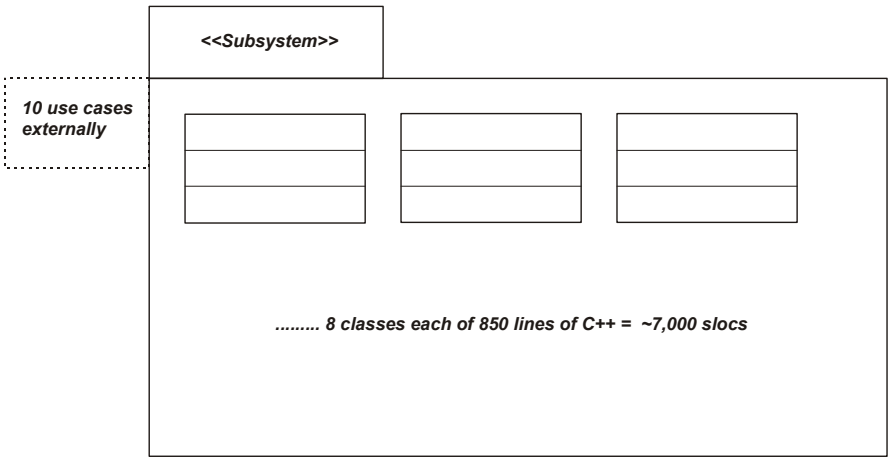
[8] Observe que isso não deve ser um limite superior inflexível; o comprimento de uma descrição de caso uso seguirá algum tipo de distribuição estatística, na qual os extremos têm uma probabilidade menor de ocorrência.

A Hierarquia de Subsistema

Isso se parece com uma hierarquia de subsistema? Aqui estão as formas ‘padrão’ simples que utilizei. Observe essas formas conceituais utilizadas para realizar um sistema. O limite do sistema real está fora de uma coleta dessas formas e a soma de casos de uso externos para cada um é o total de casos de uso externos para o sistema; portanto, um sistema real pode ter mais de dez casos de uso externos, mas o limite superior não é ilimitado, como veremos posteriormente. Observe que não é sugerido aqui que todo o desenvolvimento deva utilizar quatro níveis de caso de uso em sua descrição. Sistemas menores (<50.000 slocs) provavelmente utilizarão apenas um ou dois.

Nível 1

No Nível 1, utilizamos casos de uso realizados por classes em zero ou mais subsistemas:



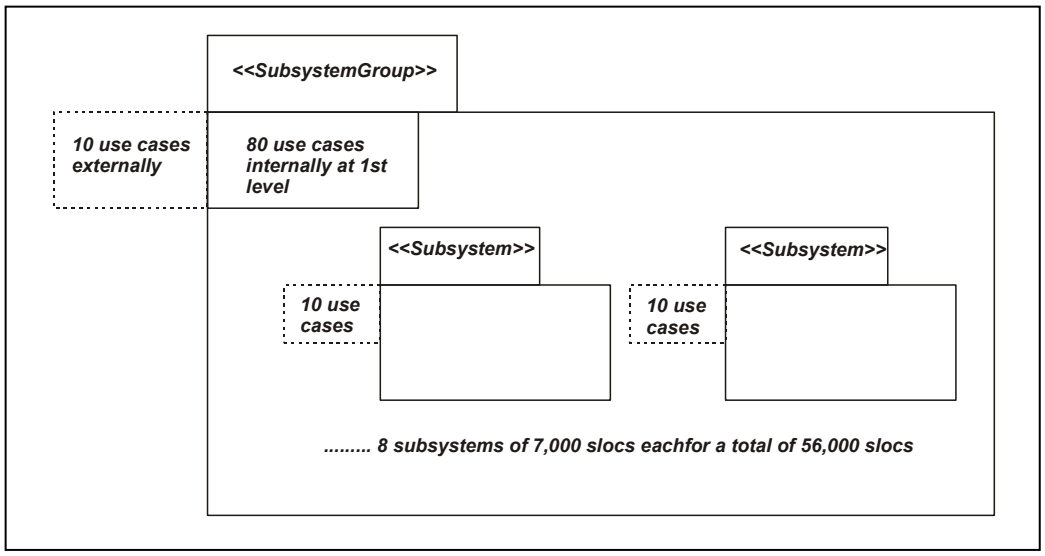
Estimando intervalos de tamanho para sistemas nesse nível (utilizando a noção de 7, mais ou menos 2):

- de 2 a 9 classes (não formadas nos subsistemas)—1700 slocs a 8000 slocs ou
- 1 subsistema de 5 classes totalizando 4000 slocs até
- 9 subsistema de 7 classes totalizando 53.550 slocs,

com os casos de uso expressos para serem realizáveis pelas instâncias de classe. É um intervalo de 2–76 casos de uso. Esses são limites vagos, pelo menos o limite superior é—a probabilidade de construir um sistema dessa maneira (neste tamanho), nunca expressando o comportamento desejado em alguma forma de nível mais alto, deve declinar para zero nesse limite. Uma contagem maior de casos de uso poderá indicar alguma patologia.

Nível 2

No próximo nível, temos um grupo de subsistemas de oito subsistemas. Acho que ele é equivalente a um CSCI (Item de Configuração do Sistema de Computador) na terminologia de defesa. Nesse nível, os casos de uso são realizados por colaborações de subsistemas:



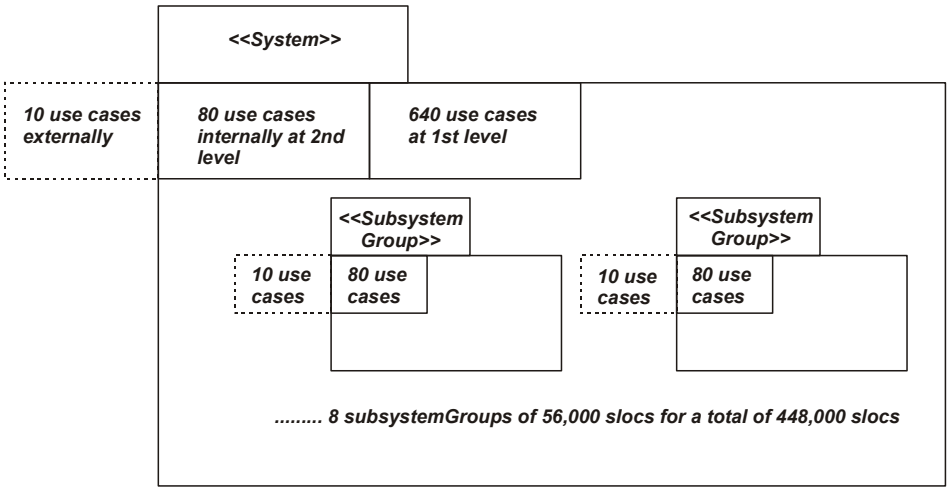
Estimando intervalos de tamanho para sistemas nesse nível (utilizando a noção de 7, mais ou menos 2):

- de 1 subsystemGroup de 5 subsistemas de 5 classes totalizando 22.000 slocs, para
- 9 subsystemGroups de 7 subsistemas cada de 7 classes, totalizando 370.000 slocs

É um intervalo de 4–66 casos de uso externos. Novamente, esses limites são vagos.

Nível 3

Neste próximo nível, temos um sistema (de grupos de subsistemas). No Nível 3, os casos de uso são realizados por colaborações de grupos de sistemas:



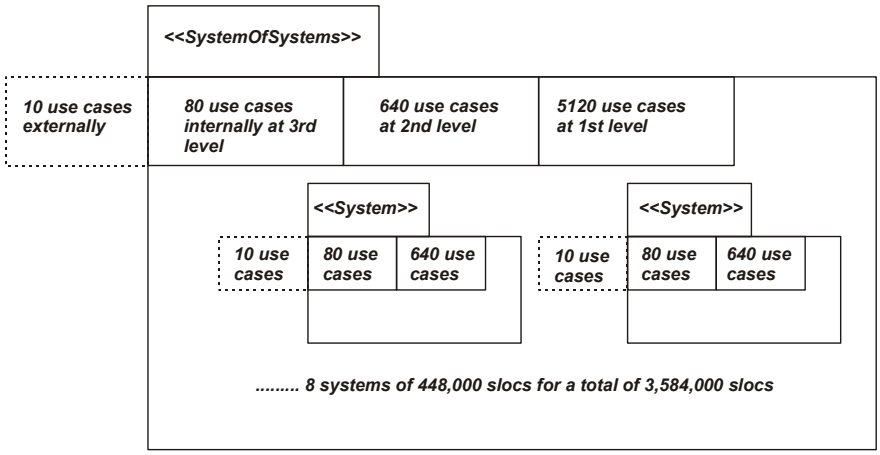
Estimando intervalos de tamanho para sistemas nesse nível (utilizando a noção de 7, mais ou menos 2):

- de 1 sistema de 5 subsystemGroups de 5 subsistemas de 5 classes totalizando 110.000 slocs a
- 9 sistemas de 7 subsystemGroups cada de sete subsistemas cada de sete classes, totalizando 2.600.000 slocs.

É um intervalo de 3–58 casos de uso externos. Novamente, esses limites são vagos.

Nível 4

No próximo nível, temos um sistema de sistemas. No Nível 4, os casos de uso são realizados por colaborações de sistemas:



Estimando intervalos de tamanho para sistemas nesse nível (utilizando a noção de 7, mais ou menos 2):

- de 1 sistema de sistemas de 5 sistemas de 5 subsystemGroups de 5 subsistemas de 5 classes totalizando 540.000 slocs, para
- 9 sistemas de sistemas de 7 sistemas cada de 7 subsystemGroups cada de 7 subsistemas cada de 7 classes, totalizando 18.000.000 slocs

É um intervalo de 2–51 casos de uso externos. Novamente, esses limites são vagos. Suponho que sejam possíveis agregados maiores, mas não quero pensar neles!

Esforço por Caso de Uso

Podemos obter alguma percepção para o esforço de caso de uso, estimando o esforço para esses tamanhos nominais em cada um dos níveis. Utilizando a ferramenta Estimate Professional™ [9] (baseada nos modelos[10] SLIM de COCOMO 2 e[11] Putnam), definir a linguagem para C++ (outros condutores de custo definidos para nominal) e calcular o esforço para cada um dos tipos de sistema de exemplo em cada ponto de tamanho nominal (assumindo 10 casos de uso externos), fornece os resultados encontrados na Tabela 1.

Tamanho (slocs)	Horas de esforço/sistema de negócios simples de caso de uso	Horas de esforço/sistema científico de caso de uso	Horas de esforço/sistema complexo de comando e controle de caso de uso
7000 (L1)	55 (intervalo 40-75)	120 (intervalo 90-160)	260 (intervalo 190-350)
56000 (L2)	820 (intervalo 710-950)	1700 (intervalo 1500-2000)	3300 (intervalo 2900-3900)
448000 (L3)	12000	21000	38000
3584000 (L4)	148000	252000	432000

[9] A Software Productivity Center Inc, http://www.spc.ca/ fornece a ferramenta Estimate Professional.

[10] Consulte Boehm81 e http://sunset.usc.edu/COCOMOII/cocomo.html.

[11] Consulte Putnam92.

Tabela 1: Esforço por Caso de Uso para Vários Tipos de Amostra

Os intervalos mostrados na Tabela 1 para o Nível 1 (L1) e Nível 2 (L2) levam em conta a complexidade de um caso de uso individual—estimado por analogia com a matriz de complexidade de código do COCOMO. No L2, acredito que a variação com a complexidade iniciará para ser incluída na caracterização pelo tipo de sistema, para que o caso de uso de um sistema complexo de comando e controle de nível mais alto contenha uma mistura de complexidades em um nível mais baixo. A plotagem disso em uma escala log-log produz a Figura 2.

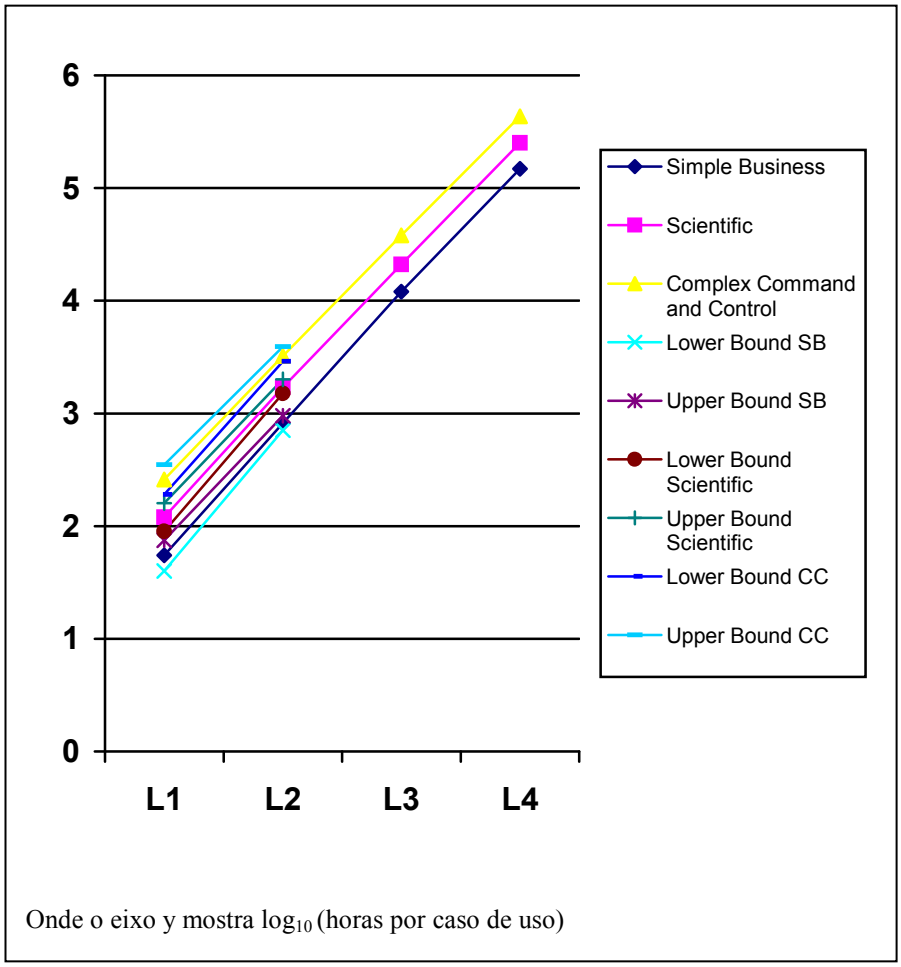


Figura 2: Esforço de Caso de Uso por Tamanho

A partir disso pode ser visto que o número antigo de Objectory de 150–350 horas/caso de uso ($10^{2,17}$–$10^{2,54}$) ajusta-se bem no L1, isto é, esses são casos de uso que podem ser realizados com colaborações de classes—portanto, no final, há alguma justificativa para esse número. No entanto, não é adequado caracterizar todos os projetos durante a análise—como um colega disse em uma comunicação de e-mail, “fica muito ‘uniforme’”.

Estimativa de Esforço

Agora, sistemas reais não se ajustarão a essas fendas convenientes, portanto, para ajudar a deduzir como um sistema deve ser caracterizado, podemos utilizar os limites vagos derivados ao longo do caminho e plotá-los conforme ilustrado na Figura 3.

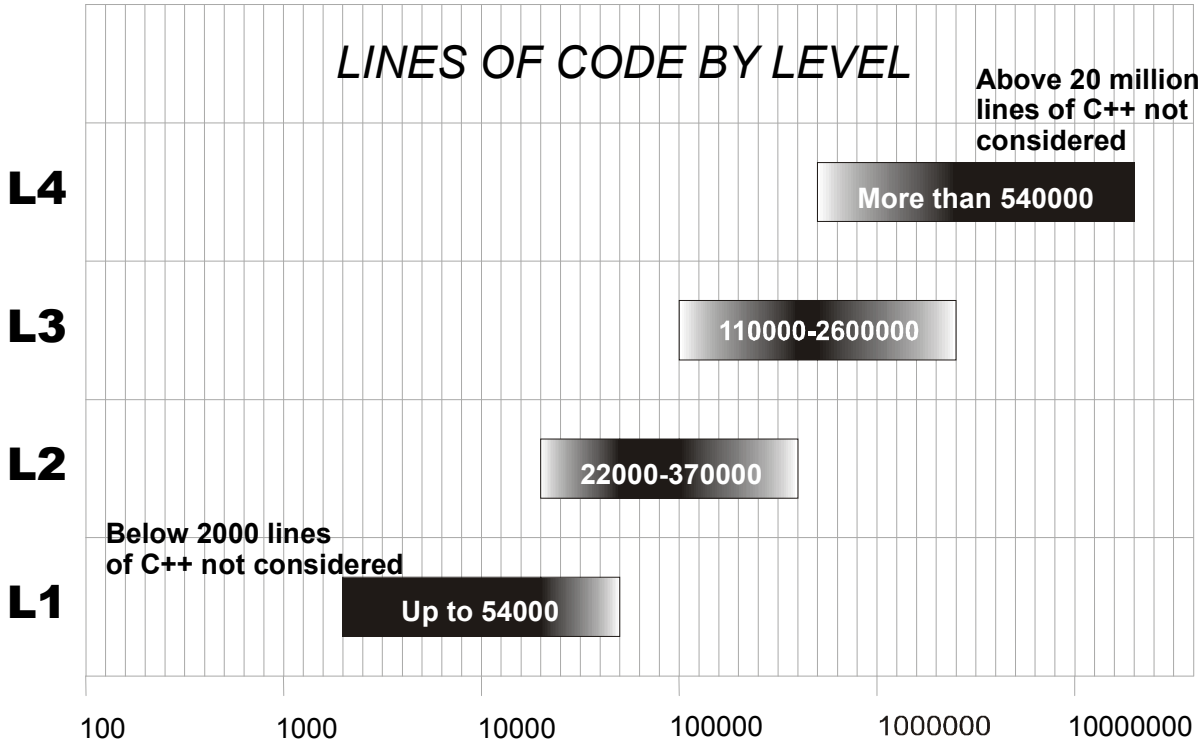


Figura 3: Dimensionar Faixas para Cada Nível

Na Figura 3, vemos que sistemas de até 22000 slocs são mais prováveis de serem descritos no Nível 1, com uma contagem de casos de uso entre 2–30. Contagens de caso de uso mais altas neste tamanho podem indicar que a granularidade dos casos de uso é muito fina.

Entre 22000 e 54000 slocs, poderá haver uma mistura de casos de uso dos Níveis 1 e 2, com uma contagem de casos de uso entre 4 (todos no Nível 2) e 76 (todos no Nível 1). Como o gráfico tenta mostrar, esses valores extremos têm uma baixa probabilidade.

Entre 54000 e 110000 slocs, é possível que um sistema bem estruturado possa ser descrito inteiramente no Nível 2, com uma contagem de casos de uso entre 10 e 20; a mistura poderá ser L1/L2/L3 (1–160 casos de uso, com esses extremos tendo uma probabilidade ***extremamente*** baixa).

Entre 110000 e 370000 slocs, existe possivelmente uma mistura do Nível 2 e Nível 3, com uma contagem de casos de uso entre 3 (todos no Nível 3) e 66 (todos no Nível 2).

Entre 370000 e 540000 slocs, se descritos inteiramente no Nível 3, poderá haver uma contagem de casos de uso entre 9 e 12; a mistura poderá ser L2/L3/L4 (1–100 casos de uso, com esses extremos tendo uma probabilidade ***extremamente*** baixa).

Entre 540000 e 2600000 slocs, existe possivelmente uma mistura do Nível 3 e Nível 4, com uma contagem de casos de uso entre 2 (todos no Nível 4) e 60 (todos no Nível 3).

Acima de 2600000 slocs, a contagem de casos de uso no Nível 4 deve subir de ~8.

Quantos Casos de Uso São Suficientes?

Algumas observações interessantes fluem disso que suportam alguns métodos empíricos. A pergunta é feita com freqüência: "Qual é o número de casos de uso que indique que há muitos?" Essa pergunta normalmente indica a quantidade excessiva ***durante a captura de requisitos.*** A resposta parece ser que mais de ~70, mesmo para o sistema maior, possivelmente indica uma granularidade bem fina antes do design. Entre 5–40 é confortável, mas o número em si, sem consideração do nível, não pode ser utilizado para estimar tamanho e esforço. Esse é o número ***inicial***, adequado para um nível específico. As centenas de contagens de casos de uso virão se um supersistema grande for decomposto em sistemas e, em seguida, em subsistemas e assim por diante. Se os casos de uso foram desenvolvidos até o nível de classe ser atingido, então, a contagem final poderá ser de centenas ou até milhares (digamos, ~600 para um projeto de 140 equipe-ano ou algo como 15 pontos de função por caso de uso). No entanto, isso não ocorrerá como uma decomposição pura de caso de uso, independente do design. Esses casos de uso surgem do processo descrito em Jacobson97—no qual os casos de uso em um nível do sistema são particionados para o comportamento alocado através de subsistemas, para o qual os casos de uso de nível mais baixo podem ser escritos (com outros subsistemas como agentes).

Procedimento de Estimativa de Esforços

Portanto, como procedemos para fazer uma estimativa? Existem alguns pré-requisitos: uma estimativa baseada em casos de uso não pode ser feita sem algum entendimento do problema do domínio e sem ***já ter uma idéia do tamanho proposto do sistema e algumas idéias da arquitetura, adequadas para o estágio no qual a estimativa está sendo feita.***

Esse primeiro corte grosseiro em uma estimativa pode ser feito utilizando a opinião de um especialista ou um pouco mais formalmente através da técnica de Wideband Delphi (isso foi inventado pela organização Rand em 1948, consulte Boehm81 para obter uma descrição). Isso permitirá que o estimador coloque o sistema em uma das faixas de tamanho na Figura 3. Essa colocação irá sugerir um intervalo para a contagem de casos de uso e indicará o nível da expressão (L1, L1/L2 e assim por diante). O estimador deve, então, decidir com base no conhecimento atual da arquitetura e do vocabulário do domínio, se os casos de uso se ajustam adequadamente em um nível, são divididos distintamente ou são uma mistura de níveis (na forma em que o fluxo de eventos é expresso).

A partir dessas considerações também deve ficar aparente se os dados são possivelmente patológicos; por exemplo, se a estimativa de Delphi for 600.000 linhas de código (ou equivalente em pontos de função) e tiver havido pouco trabalho arquitetural, de forma que ainda não se sabe muito sobre a estrutura do sistema, a Figura 3 sugere que a contagem de casos de uso deve ficar entre 2 (todos no Nível 4) e 14 (todos no Nível 3). Se a contagem de casos de uso for realmente 100, então, os casos de uso poderão ter sido prematuramente decompostos ou a estimativa de Delphi estará bem distante.

Continuando nesse exemplo: se a contagem real de casos de uso for 20 e o estimador decidir que eles são todos do L3 e, ainda mais, que o comprimento do caso de uso for de 7 páginas em média e o sistema for do tipo de negócio complexo, então, as horas por caso de uso (da Figura 2) serão 20.000. Isso deve ser multiplicado por 7/9 para considerar a complexidade mais baixa aparente (baseado no comprimento de caso de uso). Portanto, o esforço total por esse meio é 20*20000*(7/9) = ~310.000 equipe-horas ou 2050 equipe-meses. De acordo com Estimate Professional, 600.000 linhas de código C++, para um sistema de negócios complexo, requer 1928 equipe-meses. Portanto, nesse exemplo inventado há um bom acordo.

Se a contagem real de casos de uso era 5 e o estimador decidir que eles devem ser divididos, 1 em L4 e 4 no nível 3, e, ainda mais, que o caso de uso em L4 tem 12 páginas e os casos de uso em L3 têm em média 10 páginas, então, o esforço será 1*250.000*12/9+4*21000*(10/9) = ~2800 equipe-meses. Isso parece sugerir que a estimativa de Delphi talvez precise ser revisitada, embora dado que uma parte principal do sistema ainda é entendida apenas em um nível muito alto, de qualquer forma os limites de erro serão maiores.

Se a estimativa original de Delphi era de 100.000 linhas de C++, a indicação da Figura 3 é que os casos de uso devem estar no L2 e que deve haver cerca de 18 deles. Se realmente houver 20, como no primeiro exemplo, a aplicação do método sem considerar o nível real do caso de uso fornecerá um resultado inválido, se a estimativa de Delphi estiver errada.

O estimador deve verificar, portanto, se os casos de uso estão realmente no nível de abstração sugerido (L2) e podem ser realizados por uma colaboração de subsistemas e se os casos de uso não estão todos realmente em L3—embora o método de Wideband Delphi normalmente não seja tão ruim (isto é, um prognóstico de 100.000 quando o real está próximo a 600.000). No entanto, o ponto é que esse método de estimativa não pode continuar com confiança sem a construção de alguma arquitetura imaginária ou conceitual, que se alinha com o nível de caso de uso. Para um estimador muito experiente no domínio, o modelo poderá ser mental permitindo que seja feito um julgamento do nível; para um estimador e uma equipe menos experientes, é melhor fazer alguma modelagem arquitetural para ver como os casos de uso podem ser bem realizados em um nível específico.

A contagem para um caso de uso de expressão mista (ou seja, uma mistura do Nível N e do Nível N+1) deve ser contada como $n=8^{(\text{distância fracional entre os dois níveis})}$ do tipo de caso de uso de limite mais baixo. Portanto, um caso de uso avaliado em 50% L1 e 50% L2 deve ser contado como $8^{0,5} = 3$ casos de uso de L1 para obter a contagem global. Um caso de uso avaliado em 30% entre L2 e L3 deve ser contado como $8^{0,3}$ casos de uso de L2 = 2 casos de uso de L2. Um caso de uso avaliado em 90% do caminho entre L2 e L3 deve ser contado como $8^{0,9} = 7$ casos de uso de L2.

Ajuste de Tamanho da Tabela

Na verdade, existe um ajuste adicional que precisa ser feito nos números de horas/uso individuais para levar em conta o tamanho global—os números de esforço são adequados em cada nível ***no contexto de sistemas desse tamanho.*** Portanto, em L1, na Tabela 1, 55 horas por caso de uso serão aplicadas ao construir um sistema de 7000 slocs. O número real dependerá do tamanho total do sistema, portanto, se o sistema a ser construído tiver, digamos, 40.000 slocs e houver 57 casos de uso no Nível 1 descrevendo-o, o esforço não será 55*57 horas para um sistema de negócio simples, mas $(40/7)^{0,11} * 55 = 66$ horas/caso de uso. Isso baseia-se no relacionamento de COCOMO 2 do tamanho do esforço. De acordo com o modelo de COCOMO, Esforço = A * $(\text{Tamanho})^{1,11}$, em que:

- Tamanho é ksloc
- A terá condutores de custo recriados em
- Os fatores de escala do projeto são nominais (fornecendo 1,11 para o expoente)

Observe que esses cálculos podem ser recriados em uma ferramenta como o Estimate Professional para eliminar o ônus do cálculo; eles são mostrados aqui na totalidade.

Entretanto, o esforço por ksloc, ou por unidade se desejar, é igual a A* $(\text{Tamanho})^{1,11}$/Tamanho, que fornece A* $(\text{Tamanho})^{0,11}$, e a proporção de esforço/unidade no tamanho S1 para o esforço/unidade no tamanho S2 é $(S1/S2)^{0,11}$.

Além da estimativa de Delphi, o tamanho do sistema pode ser calculado aproximadamente a partir da contagem de casos de uso em vários níveis: se houver casos de uso N1 no Nível 1, N2 no Nível 2, N3 no Nível 3 e N4 no Nível 4, então, o tamanho total será [(N1/10)*7 + (N2/10)*56 + (N3/10)*448 + (N4/10)*3584] ksloc. E, portanto, podemos calcular os multiplicadores de esforço para cada um dos esforços por números de caso de uso na Tabela 1, dividindo esse tamanho total pelo tamanho de cada nível (em ksloc) mostrado na coluna um da Tabela 1.

- Então, no Nível 1 $(0,1*N1 + 0,8*N2 + 6,4*N3 + 51,2*N4)^{0,11}$
- No Nível 2 $(0,0125*N1 + 0,1*N2 + 0,8*N3 + 6,4*N4)^{0,11}$
- No Nível 3 $(0,00156*N1 + 0,0125*N2 + 0,1*N3 + 0,8*N4)^{0,11}$
- No Nível 4 $(0,00002*N1 + 0,00156*N2 + 0,0125* N3 + 0,1*N4)^{0,11}$

Claramente, no Nível 4, por exemplo, o número dos casos de uso do Nível 1 tem um efeito bem pequeno comparado ao número do Nível ou Nível 4.

Sumário

Foi apresentada estrutura para estimativa com base nos casos de uso. Para tornar a apresentação mais concreta, alguns valores foram escolhidos para os parâmetros da estrutura, que, se demonstrados, não estão ao acaso errados. Como sempre, tal conjectura deve ser testada com a realidade e os parâmetros novamente estimados conforme os dados são reunidos. A estrutura leva em conta a idéia do nível, do tamanho e da complexidade do caso de uso para categorias diferentes do sistema e não lança mão da decomposição funcional com granularidade fina. Para facilitar o ônus do cálculo, é possível construir um front-end para uma ferramenta como o Estimate Professional, que fornece um método alternativo de entrada de tamanho, com base em casos de uso.

Para obter comentários e feedback sobre este white paper, contate John Smith, jsmith@rational.com.

Referências

1. Armour96: Experiences Measuring Object Oriented System Size with Use Cases, F. Armour, B. Catherwood, et al., Proc. ESCOM, Wilmslow, UK, 1996
2. Boehm81: Software Engineering Economics, Barry W. Boehm, Prentice-Hall, 1981
3. Booch98: The Unified Modeling Language User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, Addison-Wesley, 1998
4. Cockburn97: Structuring Use Cases with Goals, Alistair Cockburn, Journal of Object-Oriented Programming, Set-Out 1997 e Nov-Dez 1997
5. Douglass99: Doing Hard Time, Bruce Powel Douglass, Addison Wesley, 1999
6. Fetcke97: Mapping the OO-Jacobson Approach into Function Point Analysis, T. Fetcke, A. Abran, et al., Proc. TOOLS USA 97, Santa Barbara, California, 1997
7. Graham95: Migrating to Object Technology, Ian Graham, Addison-Wesley, 1995
8. Graham98: Requirements Engineering and Rapid Development, Ian Graham, Addison-Wesley, 1998
9. Henderson-Sellers96: Object-Oriented Metrics, Brian Henderson-Sellers, Prentice Hall, 1996
10. Hurlbut97: A Survey of Approaches For Describing and Formalizing Use Cases, Russell R. Hurlbut, Technical Report: XPT-TR-97-03, http://www.iit.edu/~rhurlbut/xpt-tr-97-03.pdf
11. Jacobson97: Software Reuse – Architecture, Process and Organization for Business Success, Ivar Jacobson, Martin Griss, Patrik Jonsson, Addison-Wesley/ACM Press, 1997
12. Jones91: Applied Software Measurement, Capers Jones, McGraw-Hill, 1991
13. Karner93: Use Case Points - Resource Estimation for Objectory Projects, Gustav Karner, Objective Systems SF AB (direitos autorais da Rational Software), 1993
14. Lorentz94: Object-Oriented Software Metrics, Mark Lorentz, Jeff Kidd, Prentice Hall, 1994
15. Major98: A Qualitative Analysis of Two Requirements Capturing Techniques for Estimating the Size of Object-Oriented Software Projects, Melissa Major and John D. McGregor, Dept. of Computer Science Technical Report 98-002, Clemson University, 1998
16. Minkiewicz96: Estimating Size for Object-Oriented Software, Arlene F. Minkiewicz, http://www.pricesystems.com/foresight/arlepops.htm, 1996
17. Pehrson96: Software Development for the Boeing 777, Ron J. Pehrson, CrossTalk, January 1996
18. Putnam92: Measures for Excellence, Lawrence H. Putnam, Ware Myers, Yourdon Press, 1992
19. Rechtin91: Systems Architecting, Creating & Building Complex Systems, E. Rechtin, Prentice-Hall, 1991
20. Royce98: Software Project Management, Walker Royce, Addison Wesley, 1998
21. RUP99: Rational Unified Process, Rational Software, 1999
22. Stevens98: Systems Engineering – Coping with Complexity, R. Stevens, P. Brook, et al., Prentice Hall, 1998
23. Thomson94: Project Estimation Using an Adaptation of Function Points and Use Cases for OO Projects, N. Thomson, R. Johnson, et al., Proc. Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA '94, 1994

Rational®
the software development company

Duas Sedes:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Sem custo: (800) 728-1212
E-mail: info@rational.com
Web: www.rational.com
Localização Internacional: www.rational.com/worldwide