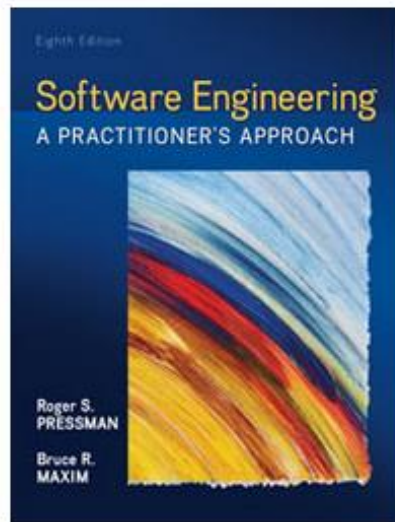


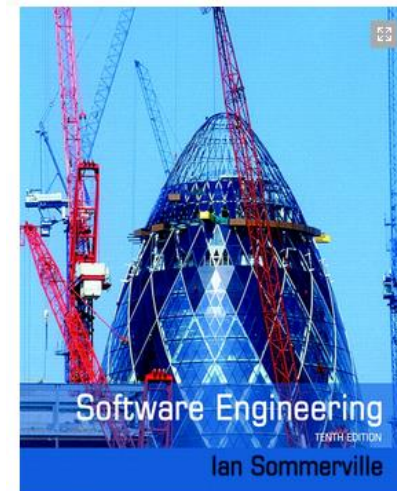


Bibliografia

- **Software Engineering – A Practitioner's Approach – Roger S. Pressman – Eight Edition – 2014**
- **Software Engineering – Ian Sommerville – 10th edition - 2015**
- Engenharia de Software – Uma abordagem profissional – Roger Pressman - McGraw Hill, Sétima Edição - 2011
- Engenharia de Software – Ian Sommerville – Nona Edição – Addison Wesley, 2007



Software Engineering: A
Practitioner's Approach, 8/e





Medição e Melhoria Contínua

- ⊕ A medição nos permite obter o entendimento do processo e do projeto, dando-nos um mecanismo para avaliação objetiva;
- ⊕ “Quando você pode medir o que você está falando em números, você sabe alguma coisa sobre aquilo; mas quando você não pode medir, quando você não pode expressar em números, o seu conhecimento é algo escasso e insatisfatório”;
- ⊕ Medições podem ser aplicadas ao processo de software com a intenção de melhoria contínua.





-



Medidas de Software

- ⊕ As medidas no mundo físico podem ser classificadas de duas formas: medidas diretas (por exemplo, o comprimento de um parafuso) e medidas indiretas (por exemplo, a “qualidade” dos parafusos produzidos , medida contando-se os parafusos rejeitados no processo de fabricação;
- ⊕ Da mesma forma, métricas de software podem ser classificadas em medidas diretas incluem quantidade de linhas de código (lines of code – LOC) produzidas, velocidade de execução ou quantidade de memória alocada.
- ⊕ Medidas indiretas de software incluem características de qualidade, tais como: eficiência, confiabilidade, usabilidade, portabilidade, facilidade de manutenção, etc.

// Comment
/* description
of code */ 





Métricas orientadas a tamanho

- ✦ Consideram o tamanho do software que foi produzido;
- ✦ Uma organização pode manter registros simples em uma tabela de medidas orientadas a tamanho.

Projeto	LOC	Esforço	\$ (000)	Pág. Doc.	Defeitos	Falhas	Pessoas Desenv.
Alfa	15000	24	180	380	130	40	5
Beta	28000	49	450	1300	290	95	8
Gama	20000	45	380	1100	250	85	7





Métricas orientadas a tamanho

- ✦ Geralmente adota-se o número de linhas de código como um valor de normalização;
- ✦ A partir dos dados rudimentares contidos na tabela, pode-se desenvolver um conjunto de métricas simples orientadas a tamanho para cada projeto:
 - Erros por cada mil linhas de código (**KLOC**);
 - Defeitos por cada mil linhas de código (**KLOC**);
 - \$ por cada mil linhas de código (**KLOC**);
 - Páginas de documentação para cada mil linhas de código (**KLOC**).

Projeto	LOC	Esforço	\$ (000)	Pág. Doc.	Defeitos	Falhas	Pessoas Desenv.
Alfa	15000	24	180	380	130	40	5
Beta	28000	49	450	1300	290	95	8
Gama	20000	45	380	1100	250	85	7



Métricas orientadas a tamanho – Observações

- ⊕ Métricas orientadas a tamanho **NÃO** são aceitas universalmente como a melhor maneira de se medir os processos de software;
- ⊕ A maior parte da controvérsia gira em torno do uso de linhas de código como medida principal;
- ⊕ Alguns argumentam que as medidas LOC são fortemente dependentes da Linguagem de Programação.



Projeto	LOC	Esforço	\$ (000)	Pág. Doc.	Defeitos	Falhas	Pessoas Desenv.
Alfa	15000	24	180	380	130	40	5
Beta	28000	49	450	1300	290	95	8
Gama	20000	45	380	1100	250	85	7



Métricas orientadas a Função



- ✦ Usam uma medida de **funcionalidade** fornecida pela aplicação como um valor de normalização;
- ✦ A métrica orientada a função mais largamente aceita é a **FP – Function Point**, conforme visto no capítulo anterior;
- ✦ Os proponentes argumentam que são vantajosas por serem independentes da Linguagem de Programação, tornando-a mais atraente como abordagem de estimativas;
- ✦ No entanto, oponentes argumentam que o método baseia-se em dados **subjetivos** ao invés de objetivos e que a contagem do domínio de informações podem ser difícil de ser coletada.





Reconciliando métricas LOC e FP

- ⊕ Muitos estudos já tentaram relacionar medidas **LOC** e **FP**;
- ⊕ A tabela abaixo exibe estimativas aproximadas da média do número de linhas de código necessárias para criar um ponto de função em várias linguagens de programação:

Linguagem de Programação	LOC médio por Ponto de Função
Assembler	119
C	97
Cobol	61
C#	54
Java	53
C++	50
JavaScript	47

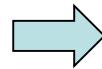
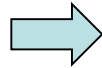
Fonte: www.qsm.com





Reconciliando métricas LOC e FP

- ⊕ Uma LOC de **C++** fornece aproximadamente 2 vezes a “funcionalidade” (em média) de uma LOC de **C**;
- ⊕ Usando-se essa tabela, pode-se **retroagir** o software existente para estimar o número de pontos de função, uma vez conhecido o número total de instruções da Linguagem de Programação.



Linguagem de Programação	LOC médio por Ponto de Função
Assembler	119
C	97
Cobol	61
C#	54
Java	53
C++	50
JavaScript	47

Fonte: www.qsm.com





Métricas LOC e FP – Considerações



- ⊕ Descobriu-se que métricas baseadas em Pontos de Função e LOC são indicadores relativamente precisos de trabalho e custo de desenvolvimento de software.
- ⊕ No entanto, para usar LOC e FP para estimativas, deve ser estabelecida uma referência histórica de informações.
- ⊕ Assim, as seguintes questões são importantes para desenvolvimento de software com qualidade e produtividade:
 - Qual foi a produtividade do desenvolvimento de software nos projetos passados?
 - Qual foi a qualidade do software produzido?
 - Como os dados de produtividade e qualidade do passado podem ser extrapolados para o presente?
 - Como isso pode nos ajudar a melhorar o processo e planejar novos projetos mais precisamente?





Métricas Orientadas a Objeto



- ✓ **Número de Scripts de Cenário**: Análogo aos casos de usos, representam a sequência detalhada de passos que descrevem a interação entre usuário e sistema. O número de scripts está correlacionado ao tamanho da aplicação e com o número de casos de teste que devem ser desenvolvidos para exercitar o sistema;
- ✓ **Número de Classes-Chave**: Correspondem aos componentes altamente independentes que são definidos na modelagem de domínio (análise orientada a objetos). São as classes essenciais ao domínio do problema. A quantidade dessas classes é uma indicação do esforço necessário para se desenvolver o software.
- ✓ **Número médio de classes de apoio**: São as classes necessárias para se implementar o sistema, mas não estão intimamente relacionadas ao domínio do problema. Exemplo: classes de interface, classes de acesso e manipulação de banco de dados, cálculos, etc. A quantidade dessas classes é uma indicação do esforço necessário para se desenvolver o software.



Métricas Orientadas a Casos de Uso



- ✓ Casos de uso são amplamente usados como método para descrever requisitos no nível de domínio do negócio, sugerindo características e funções do software;
- ✓ São independentes de Linguagem de Programação e diretamente proporcional ao tamanho do software e ao número de casos de teste que deverão ser projetados para exercitar a aplicação;
- ✓ No entanto, podem ser criados em muitos níveis de abstração, dificultando a sua utilização como medida de normalização.



Métricas dentro do Processo de Software



- ⊕ A maioria dos desenvolvedores não mede **NADA**;
- ⊕ Infelizmente, poucos têm vontade de começar;
- ⊕ O problema, de certa forma, é cultural;
- ⊕ Tentar coletar medidas onde nenhuma medida foi coletada no passado, muitas vezes causa uma resistência;
- ⊕ “Por que precisamos fazer isso?” Pergunta o gerente do projeto apressado. “Não vejo razão para isso”, concorda um programador muito atarefado.





Estimativas de Projeto de Software

- ✓ O gerenciamento de projeto de software começa com uma série de atividades chamadas coletivamente de **Planejamento de Projeto**;
- ✓ Antes de se iniciar o projeto, a equipe de software deverá fazer uma **estimativa do trabalho**, os **recursos** que serão necessários e o **tempo** necessário para a sua conclusão;
- ✓ Uma vez executadas essas tarefas, a equipe de projeto deverá estabelecer um **cronograma** que defina as **tarefas** de Engenharia de Software e as metas, deverá identificar os responsáveis pela execução de cada tarefa e especificar as **dependências** entre as tarefas que podem ter influência no progresso do trabalho.





Qual o impacto da falta de planejamento?





Impacto da falta de planejamento(*)

- ✓ A falta de planejamento é uma das falhas mais **críticas** que um projeto pode ter;
- ✓ O planejamento **eficaz** é necessário para se resolver problemas **precoces** (no início do projeto) com **baixo custo**, em vez de problemas **tardios** (que surgem mais tarde no projeto) **com alto custo**;
- ✓ Os projetos consomem em média 80% do tempo em **retrabalho** – corrigindo erros que foram feitos no início do projeto.



(*) Steve McConnell – Sobrevivência de projeto de software, 1998



Estimativas de Projeto de Software

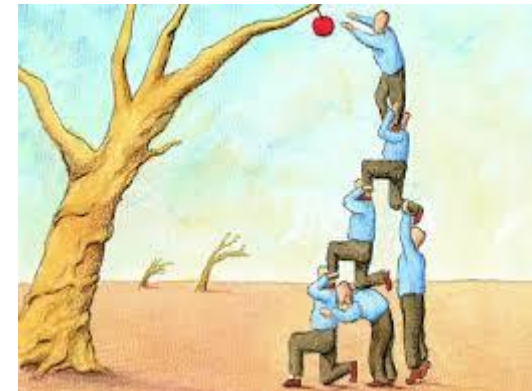
- ✦ Embora estimar seja muito mais arte do que ciência, não precisa ser conduzida de forma aleatória;
- ✦ Existem técnicas úteis para se estimar tempo e esforço;
- ✦ As métricas de projeto e processo podem proporcionar perspectivas históricas e valiosas informações para se gerar estimativas quantitativas;
- ✦ A experiência de todos os envolvidos no projeto pode também auxiliar à medida em que as estimativas são desenvolvidas e revisadas;
- ✦ Estimativas trazem risco inerente e esse risco leva à incerteza.





Experiência da equipe

- ⊕ Alguém que desenvolve pela **primeira vez** uma sofisticada aplicação de comércio eletrônico, pode considera-la excessivamente complexa;
- ⊕ No entanto, uma equipe de Engenharia de Software para Web desenvolvendo sua **décima aplicação** Web para comércio eletrônico, consideraria isso como um **trabalho comum**. Nesse caso, as estimativas seriam bem mais realistas.





Informações históricas

- ⊕ A disponibilidade de informações históricas tem uma forte influência sobre o risco das estimativas;
- ⊕ Empregar procedimentos que funcionaram pode melhorar áreas problemáticas;
- ⊕ Quando há métricas de software abrangentes para projetos passados, as estimativas podem ser feitas com maior segurança, podem ser estabelecidos os cronogramas para evitar dificuldades passadas, e o risco em geral é reduzido.





Mas, e quando os requisitos mudam?

ou

E quando o escopo do projeto é mal entendido?





Risco da Estimativa

- ✦ O **risco** das estimativas é medido pelo grau de **incerteza** nas estimativas quantitativas estabelecidas para os **recursos**, **custo** e **cronograma**;
- ✦ Se o **escopo** do projeto é **mal entendido** ou se os **requisitos** do projeto sofrem **alterações**, a incerteza e o risco das estimativas tornam-se **perigosamente altos**;
- ✦ Abordagens modernas de Engenharia de Software assumem uma **visão iterativa e evolucionária** de desenvolvimento do software. Em tais abordagens, embora nem sempre politicamente aceitável, deve-se voltar à estimativa (à medida em que mais informações são conhecidas) e **revisá-la** quando o cliente fizer alterações nos requisitos.





Processo de Planejamento de Projeto – Tarefas

- Estabeleça o escopo do projeto;
- Determine a viabilidade;
- Analise os riscos;
- Defina os recursos necessários;
- Estime o custo e a mão de obra;
- Desenvolva um cronograma de projeto.





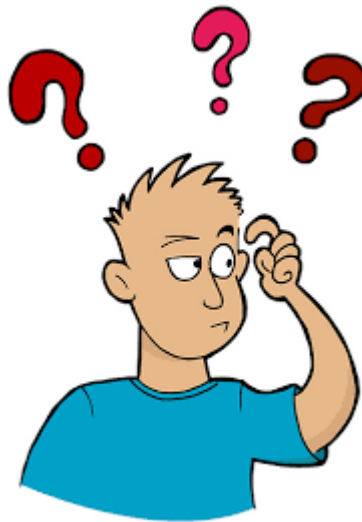
Escopo e Viabilidade do Software

- O escopo do software descreve as funções e características que devem ser fornecidas aos usuário finais;
- Descreve também os dados que entram e saem;
- Descreve também o conteúdo que é apresentado aos usuários como consequência do software;
- Descreve ainda, o desempenho, restrições, interfaces e confiabilidade que limitam o sistema.





Como definir o escopo do Software ?





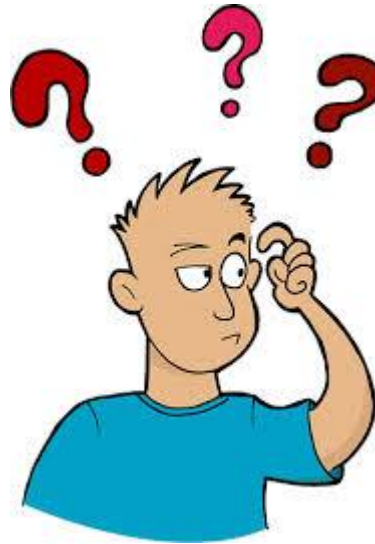
Definição do Escopo do Software

- O escopo do software pode ser definido por meio de uma narrativa (texto) desenvolvido após o levantamento do sistema junto ao usuário;
- Pode-se também empregar cenários de casos de uso;
- Funções descritas na definição do escopo são avaliadas e algumas são refinadas para se fornecer maiores detalhes antes de se iniciar as estimativas.





O que fazer após a definição do Escopo do Software?





Após a definição do Escopo do Software

✚ Algumas perguntas devem ser proferidas ao cliente:

- ✓ Podemos criar software que atenda a esse escopo?
- ✓ O projeto é viável?





Viabilidade do Software

- ⊕ **Tecnologia**. O projeto é tecnicamente viável?
- ⊕ **Finanças**. O projeto é financeiramente viável?
- ⊕ **Tempo**: O tempo que o projeto leva para atingir o mercado vence a concorrência?
- ⊕ **Recursos**: A organização tem os recursos necessários para ser bem-sucedida?





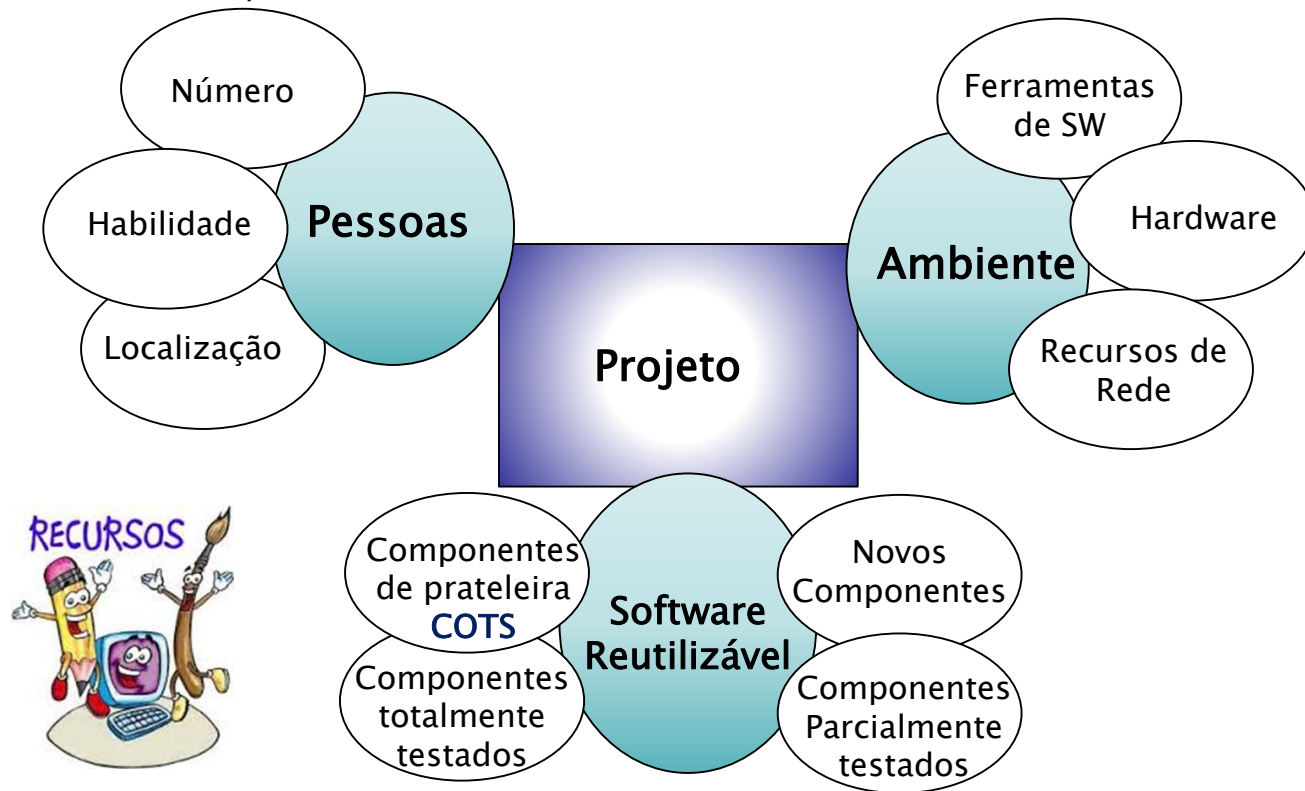
Viabilidade do Software

- ⊕ A determinação do escopo, de forma isolada, não é suficiente;
- ⊕ Uma vez entendido o escopo, deve-se trabalhar para determinar se pode ser feito segundo as dimensões: Tecnologia, Finanças, Tempo e Recursos;
- ⊕ Essa é uma parte crucial do processo de Planejamento do Projeto, que muitas vezes passa despercebida.
- ⊕ Sem essas ponderações, o projeto pode estar condenado desde o início.



Recursos

- ✚ A segunda tarefa do planejamento do software é a estimativa de recursos que são necessários para executar o trabalho de desenvolvimento do software;
- ✚ A figura abaixo mostra as três principais categorias de recursos de Engenharia de Software: pessoas, componentes de software reutilizáveis e o ambiente de desenvolvimento (hardware e ferramentas de software);

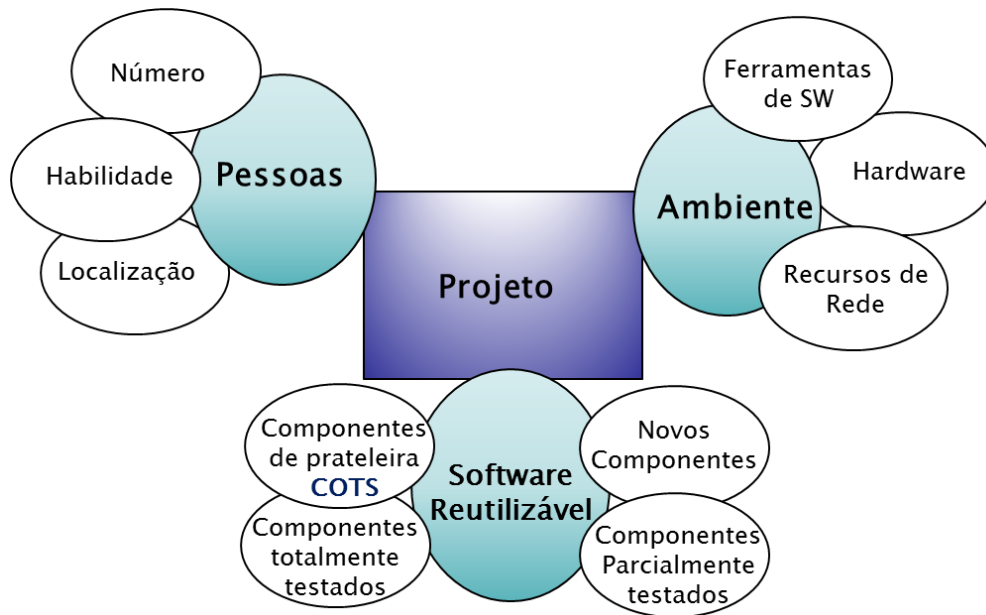


Fonte: Pressman



Recursos

- ✦ Cada recurso deve ser especificado com quatro características: descrição do recurso, definição da disponibilidade, instante em que o recurso será necessário e o tempo durante o qual o recurso será requerido.
- ✦ As duas últimas características podem ser vistas como janela de tempo. (hardware e ferramentas de software);



Fonte: Pressman

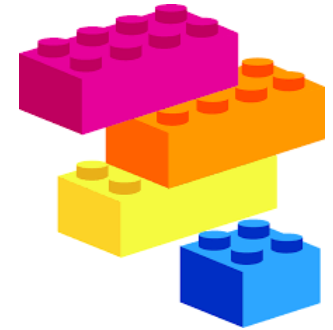


Recursos Humanos

- ✦ O gerente do projeto inicia o projeto avaliando o escopo do software e selecionando as habilidades necessárias para o desenvolvimento do software;
- ✦ São especificadas a posição organizacional (gerente, engenheiro de software, analista, etc) e a especialização (redes, banco de dados, design, etc...);
- ✦ Projetos pequenos (poucas pessoas-mês), um único profissional pode executar todas as tarefas, consultando os especialistas quando necessário;
- ✦ Para projetos maiores, a equipe pode estar geograficamente dispersa, em localizações diferentes e portanto, a localização de cada recurso é necessária;
- ✦ O número de pessoas necessárias para um projeto de software somente pode ser determinado após uma estimativa do esforço de desenvolvimento (por exemplo, pessoas-mês). As técnicas para estimativas serão apresentadas mais adiante.

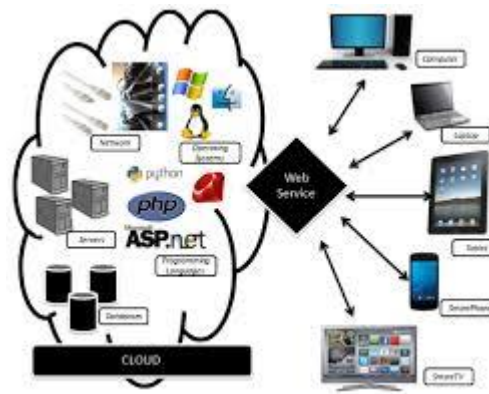


Fonte: Pressman



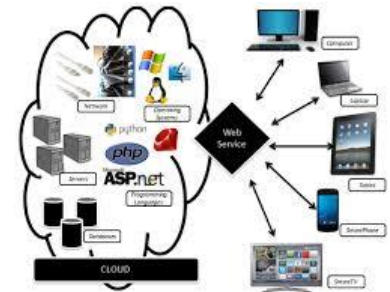
Recursos de Software reutilizáveis

- ✦ A Engenharia de Software baseada em componentes (**CBSE – Component-based Software Engineering**) enfatiza a reusabilidade, isto é a criação de blocos básicos de software.
- ✦ Esses blocos básicos, chamados de COMPONENTES, devem ser catalogados para facilitar a referência, padronizados para facilitar a aplicação e validados para facilitar a integração;





Recursos de Software reutilizáveis



⊕ Há quatro categorias de recursos de software reutilizáveis:

- **Componentes de Prateleira:** Software existente que pode ser adquirido de terceiros (COTS);
- **Componentes totalmente testados:** Componentes já desenvolvidos em projetos anteriores e que são similares ao software a ser criado no projeto atual;
- **Componentes parcialmente testados:** Componentes já existentes, mas que necessitam de grandes modificações;
- **Novos componentes:** devem ser criados pela equipe no projeto atual.



Recursos de Ambiente

- ⊕ O ambiente que suporta um projeto de software, muitas vezes chamado de SEE – Software Engineering Environment - ambiente de Engenharia de Software, incorpora hardware e software;
- ⊕ Faz parte do planejamento do projeto, verificar se esses recursos estarão disponíveis a partir do início do projeto;
- ⊕ Por exemplo: dependência de versões de software, dependência de aquisição de hardware, etc





Estimativa do Projeto de Software

- ✓ As estimativas de custo e esforço de software nunca serão uma ciência exata;
- ✓ No entanto, uma série de etapas sistemáticas podem proporcionar estimativas com riscos aceitáveis. Algumas sugestões:
 1. **Adie** a estimativa no decorrer do projeto. (Após a conclusão do projeto, pode-se obter uma estimativa com precisão de 100%). Infelizmente, esta opção, embora atrativa, **não é prática**. As estimativas sempre devem ser feitas no início do projeto;
 2. Fundamente suas estimativas em **projetos similares** que já foram completados;
 3. Use **técnicas de decomposição** relativamente simples. Assuma uma abordagem do tipo **“dividir para conquistar”** para a estimativa do projeto de software;
 4. Use um ou mais **modelos empíricos** para estimativas;

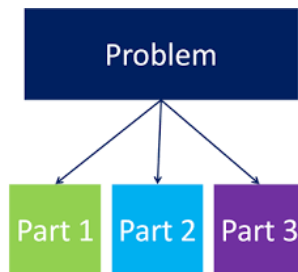




Estimativas – Técnicas de Decomposição



- ⊕ A estimativa do projeto é baseada na técnica Divide-and-Conquer;
- ⊕ O dimensionamento do software representa o primeiro grande desafio do gerente do projeto (planejador);
- ⊕ Se for adotada uma abordagem direta, o tamanho pode ser medido em linhas de código (LOC);
- ⊕ Se for adotada uma abordagem indireta, o tamanho pode ser representado por pontos de função (FP);



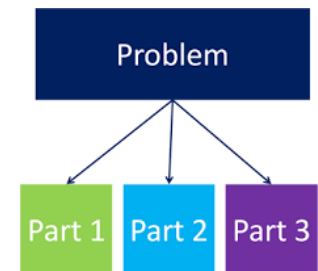
Estimativa baseada em LOC – Exemplo



- ⊕ Considere o desenvolvimento de um software **CAD** na área de Engenharia Mecânica;
- ⊕ A partir da definição do escopo do software, tenta-se decompor o software em funções de problemas que podem ser estimadas individualmente. Quanto maior a decomposição funcional, maior será a probabilidade de serem desenvolvidas estimativas LOC razoavelmente precisas;
- ⊕ Supondo-se que tenha havido uma decomposição funcional e que as principais funções do software são as descritas abaixo (subsistemas):



- ✓ Interface de usuário e recurso de controle;
- ✓ Análise Geométrica Bidimensional;
- ✓ Análise Geométrica Tridimensional;
- ✓ Gerenciamento de bases de dados;
- ✓ Recursos de Visualização da computação gráfica;
- ✓ Função de controle de periféricos;
- ✓ Módulos de controle e análise do software;

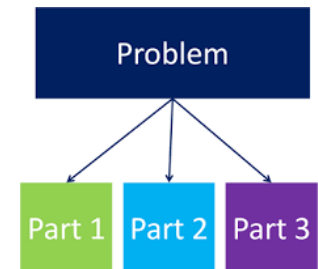


Estimativa baseada em LOC – Exemplo



- ⊕ Por meio de dados históricos ou (quando tudo o mais falhar) intuição, estimam-se valores de tamanho **otimista**, mais **provável** e **pessimista** para cada função;
- ⊕ O valor esperado pode ser calculado para a variável de estimativa (S), por meio da média ponderada entre a estimativa otimista (S_{opt}), mais provável (S_m) e pessimista (S_{pess}).
- ⊕ Por exemplo:

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6}$$

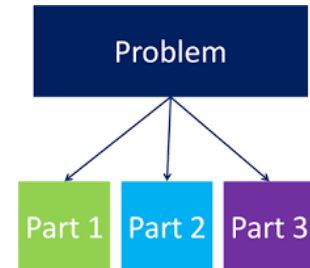




Estimativa baseada em LOC – Exemplo

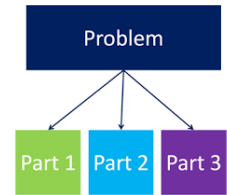


- ⊕ Para cada função é desenvolvido um intervalo de estimativas **LOC**;
- ⊕ Por exemplo, o intervalo de estimativas LOC para a função “Análise Geométrica 3-D” otimista é 4.600 **LOC**, mais provável 6.900 **LOC** e pessimista 8.600 **LOC**;
- ⊕ Aplicando-se a equação definida no slide anterior, obtém-se o resultado do valor esperado da função de análise geométrica 3D: 6.800 **LOC**;
- ⊕ As outras estimativas são obtidas de modo semelhante;





Estimativa baseada em LOC – Exemplo



✚ Desenvolvendo-se o cálculo das demais funções, obtém-se a tabela abaixo:

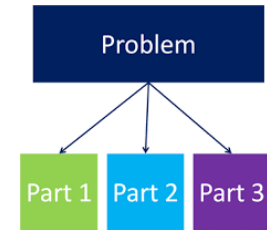
Função	LOC estimado
Interface do usuário	2.300
Análise Geométrica Bidimensional	5.300
Análise Geométrica Tridimensional	6.800
Gerenciamento de Base de dados	3.350
Recursos de Visualização	4.950
Controle de Periféricos	2.100
Módulos de controle e análise	8.400

Total de linhas de código estimadas: 33.200





Estimativas do projeto – Exemplo (LOC)



- ✦ Um exame de dados históricos indica que a produtividade média organizacional para sistemas deste tipo é de **620 LOC/pm**. (linhas de código por pessoa-mês);
- ✦ Com base em um valor bruto da mão de obra de **US\$ 8.000** por mês, o custo por cada linha de código é de aproximadamente **US\$ 13**.
- ✦ Com base na estimativa LOC e dados históricos de produtividade, o custo total estimado do projeto será de **US\$ 431.000** e o esforço estimado será **de 54 pessoas-mês**.



Estimativas do projeto – Exemplo (FP)



- ⊕ A decomposição para estimativa baseada em FP concentra-se em valores do domínio da informação;
- ⊕ Considere o mesmo projeto desenvolvimento de um software **CAD** na área de Engenharia Mecânica do exemplo anterior desenvolvido com a estimativa LOC;
- ⊕ A estimativa consiste em se determinar as entradas, saídas, consultas, arquivos e interfaces externas para o software.
- ⊕ Aplica-se a técnica de Pontos de Função, conforme apresentada no capítulo anterior.



Estimativas do projeto – Exemplo (FP)



- ⊕ Supondo-se no exemplo, que haja **20** entradas externas, **12** saídas externas, **16** consultas externas, **4** arquivos lógicos internos e **2** arquivos de interface externo;
- ⊕ Para os propósitos dessa estimativa, o fator de peso da complexidade é adotado como médio;
- ⊕ Calcula-se o fator de ajuste, conforme visto na técnica de Pontos de Função;
- ⊕ Por fim, obtém-se o número estimado de **FP = 375**;
- ⊕ Considerando-se que a produtividade média organizacional para sistemas desse tipo é **6,5** FP/pm (Function point por pessoa-mês) e que o valor bruto de mão de obra é de US\$ 8 mil por mês, pode-se obter o custo por FP aproximadamente igual a US\$ 1.230.
- ⊕ Com base na estimativa FP e nos dados históricos de produtividade, o custo total estimado do projeto será de **US\$ 461mil** e a estimativa de esforço será de **58** pessoas-mês.



Estimativa baseada em Processo

- ⊕ Nesta técnica, o processo é **decomposto** em um conjunto relativamente pequeno de tarefas, e estima-se o trabalho necessário para executar cada tarefa;
- ⊕ Assim, como as técnicas já vistas anteriormente, a estimativa baseada em processo começa com um delineamento das funções de software obtidas no escopo do projeto;
- ⊕ Para cada função, devem ser executadas uma série de **atividades estruturais**. Uma vez combinada as funções do problema e as atividades do processo, pode-se estimar o esforço (por exemplo, pessoas-mês) necessário para se executar cada atividade do processo, para cada função do software. (**Empregam-se dados históricos para se estimar o esforço**).
- ⊕ Em seguida, aplicam-se os **valores médios de preços de mão-de-obra** (custo/unidade de esforço) ao esforço estimado para cada atividade do processo.





Exemplo – Estimativa baseada em Processo

- ⊕ Consideremos o software CAD visto nos slides anteriores, no qual as funções para o desenvolvimento já foram levantadas previamente, a partir do escopo do software;
- ⊕ Pode-se desenhar uma tabela, relacionando-se funções do software com as atividades do processo, conforme slide a seguir.





Exemplo – Estimativa baseada em Processo



Fonte: Pressman

Atividade	CC	Planej.	AR	Engenharia		Liberação		AC	Totais
Tarefa ►				Análise	Design	Code	Test		
Função ▼									
UICF				0,50	2,50	0,40	5,00	n/a	8,40
2DGA				0,75	4,00	0,60	2,00	n/a	7,35
3DGA				0,50	4,00	1,00	3,00	n/a	8,50
CGDF				0,50	3,00	1,00	1,50	n/a	6,00
DBM				0,50	3,00	0,75	1,50	n/a	5,75
PCF				0,25	2,00	0,50	1,50	n/a	4,25
DAM				0,50	2,00	0,50	2,00	n/a	5,00
Totais	0,25	0,25	0,25	3,50	20,50	4,50	16,50		46,00
% Mão de Obra	1%	1%	1%	8%	45%	10%	36%		

CC=Comunicação com o cliente

AR = Análise de Risco

AC = Avaliação pelo cliente

Estimativas de esforço em pessoas-mês





Exemplo – Estimativa baseada em Processo



Fonte: Pressman



Atividade	CC	Planej.	AR	Engenharia		Liberação		AC	Totais
Tarefa ►				Análise	Design	Code	Test		
Função ▼									
UICF				0,50	2,50	0,40	5,00	n/a	8,40
2DGA				0,75	4,00	0,60	2,00	n/a	7,35
3DGA				0,50	4,00	1,00	3,00	n/a	8,50
CGDF				0,50	3,00	1,00	1,50	n/a	6,00
DBM				0,50	3,00	0,75	1,50	n/a	5,75
PCF				0,25	2,00	0,50	1,50	n/a	4,25
DAM				0,50	2,00	0,50	2,00	n/a	5,00
Totais	0,25	0,25	0,25	3,50	20,50	4,50	16,50		46,00
% Mão de Obra	1%	1%	1%	8%	45%	10%	36%		

- ⊕ A tabela relaciona as estimativas de esforço (**pessoas-mês**) para cada atividade do processo de Engenharia de Software;
- ⊕ As principais tarefas são as de **Engenharia** e de **Construção**;
- ⊕ Com base na taxa média bruta de mão de obra de **U\$8.000** por mês, o custo total estimado do projeto é de **U\$ 368.000** e o esforço estimado é de **46** pessoas-mês.



Modelos Empíricos

- ⊕ Usam fórmulas derivadas empiricamente para prever estimativas de esforço;
- ⊕ Os dados empíricos que suportam muitos modelos são obtidos de uma amostragem limitada de projetos;
- ⊕ Reflete a população de projetos com base na qual foi obtido. Portanto, é sensível ao domínio;
- ⊕ Exemplo: COCOMO – Constructive Cost Model



Modelo COConstructive COSt MOdel (COCOMO)

- ⊕ Concebido em 1981; [Boehm,1981];
- ⊕ Evoluiu para um modelo mais abrangente – **COCOMO II**; [Boehm, 2000];
- ⊕ Composto pelos seguintes sub-modelos:
 - Modelo de Composição de Aplicação: Usado nas primeiras fases da Engenharia de Software, em que protótipos das interfaces de usuário e interação de software e sistema são de suma importância.
 - Modelo do Início do Projeto: Usado quando os requisitos tiverem sido estabilizados e arquitetura básica do software estiver sido estabelecida;
 - Modelo pós-arquitetura: Usado durante a construção do software.

