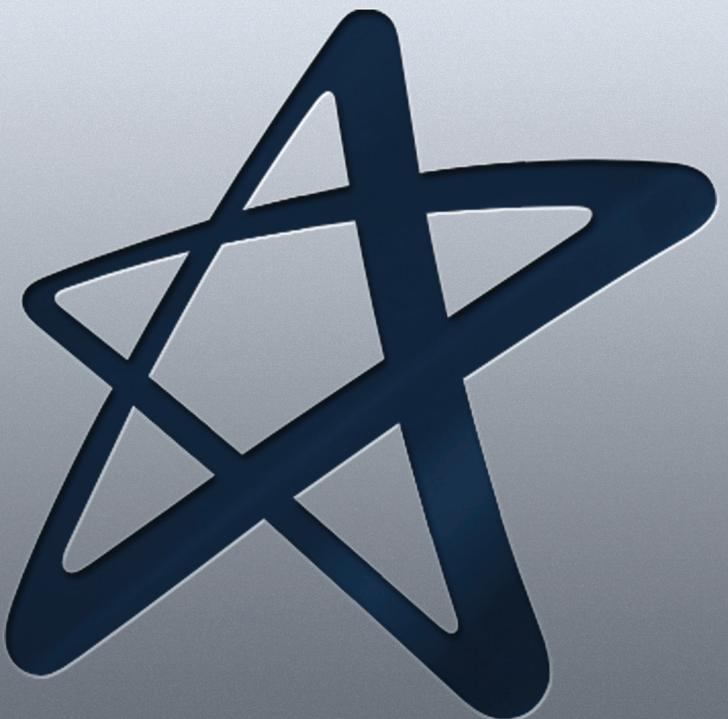


# **Engenharia de Software**



**Educação a Distância**  
Cruzeiro do Sul Educacional  
**Campus Virtual**



# Material Teórico



Ferramentas CASE e qualidade de *software*

**Responsável pelo Conteúdo:**

Prof.<sup>a</sup> Dr.<sup>a</sup> Ana Paula do Carmo Marcheti Ferraz

**Revisão Textual:**

Prof.<sup>a</sup> Me. Luciene Oliveira da Costa Santos



# UNIDADE

## Ferramentas CASE e qualidade de software



- Introdução
- Ferramentas CASE
- Qualidade de software



### Objetivo de APRENDIZADO

- Compreender as características relacionadas às ferramentas CASE.
- Refletir a importância da qualidade de software.
- Perceber a existência e as peculiaridades das técnicas e estratégias que focam a qualidade do software.
- Compreender que a qualidade do produto é dependente da qualidade do processo.
- Conhecer modelos para a melhoria da qualidade do software.

Este guia de estudo sobre a forma de organizar suas atividades para cumprir os propósitos da disciplina *online* objetiva salientar algumas informações importantes sobre esta unidade, bem como sobre o estudo *online* em si.

Organize-se também de forma a não deixar para o último dia a realização das atividades (AS e AP), pois podem ocorrer imprevistos. Encerrada a unidade, encerra-se a possibilidade de obter a nota relativa a cada atividade.

Para ampliar seu conhecimento, bem como aprofundar os assuntos discutidos, pesquise, leia e consulte os livros indicados nas Referências.



A Bibliografia Fundamental para esta Disciplina é: SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011. A Bibliografia Complementar está indicada em item específico em cada unidade.

## Contextualização

Discutiremos as questões relacionadas à produção de *software*, assim como métricas para garantir as qualidades de *software*, não são tarefas fáceis, entretanto, são fundamentais para satisfação do cliente ou dos usuários.

Nesta unidade, abordaremos as ferramentas que automatizam o trabalho do desenvolvedor de *software* durante as diversas fases do processo produtivo e até que ponto estas ferramentas interferem na produtividade da indústria do *software*, além das características de qualidade do produto e do processo de *software*, mostrando aspectos e fatores que afetam a qualidade do *software*. Além disso, você conhecerá as principais normas que visam à melhoria da qualidade de *software*.

Entender esses conceitos ajudará a compreender melhor os aspectos relacionados à Engenharia de *Software* apresentados nesta disciplina.

## Introdução



Todos os assuntos relacionados à Engenharia de *Software* estão integrados a métodos, ferramentas e procedimentos para o desenvolvimento de produtos de *software* com qualidade.

As ferramentas proporcionam um apoio automatizado ou semiautomatizado aos métodos. Quando as ferramentas são integradas de forma que a informação criada por uma ferramenta possa ser usada por outra, é estabelecido um sistema de suporte ao desenvolvimento de *software* chamado de Engenharia de *Software* Auxiliada por Computador, ou CASE, do inglês *Computer-Aided Software Engineering* (PRESSMAN, 1995).

As ferramentas proporcionam um apoio automatizado aos métodos. Hoje em dia, já existem ferramentas capazes de sustentar todas as etapas e métodos relacionados às várias fases do processo de desenvolvimento de *software* de forma automatizada.

Uma delas é a Ferramenta CASE. Nesta unidade, *não apenas apresentaremos as questões históricas e funcionais da ferramenta*, como também daremos início a conceitos relacionados à qualidade de produto de *software*.

Qualidade pode ser um conceito subjetivo – o que é qualidade para um, pode não ser para outro. Entretanto, para medirmos a qualidade de um *software*, conceitos subjetivos não podem ser considerados.

Para que possamos excluir ou definir a falta parcial ou total da qualidade num produto de *software*, é necessário que haja uma definição precisa do que é qualidade ou, pelo menos, quais são as propriedades relacionadas a partir dos princípios da Engenharia de *Software*.

Nessa linha de raciocínio é que discutiremos os pontos de qualidade, inclusive apresentando alguns conceitos e padrões definidos pelo Instituto de Engenharia de *Software* – *Software Engineering Institute* (SEI) – que, dentre tantos modelos, definiu o de certificação de qualidade de *software* – *Capability and Maturity Model Integrator* (CMMI) – que veremos, em linhas gerais, mais adiante nesta unidade.

## Ferramentas CASE



Antes de apresentarmos as características das ferramentas CASE, vamos ler algumas definições, de diferentes autores, para essas ferramentas:

- CASE refere-se a uma ampla gama de diferentes tipos de programas utilizados para apoiar as atividades do processo de *software*, como análise de requisitos, modelagem de sistema, depuração e testes (SOMMERVILLE, 2011).
- CASE é uma ferramenta ou conjunto de técnicas facilitadoras de desenvolvimento de *software* moderno, que utiliza técnicas para ajudar no trabalho dos engenheiros de *software* (REZENDE, 2005).
- Uma ferramenta CASE é um instrumento ou sistema automatizado utilizado para realizar uma tarefa da melhor maneira possível. Essa melhor maneira pode significar que a ferramenta nos torna mais precisos, eficientes e produtivos, ou que exista melhora na qualidade do produto resultante (PFLEEEGER, 2004).

De acordo com Sommerville (2011), os engenheiros fazem os produtos funcionarem, aplicando teorias, métodos e ferramentas nas situações apropriadas de modo seletivo. O *software* não é apenas um programa de computador, é, também, toda a documentação associada e os dados necessários para fazer com que esses programas funcionem corretamente.

A Engenharia de *Software* não cuida apenas do desenvolvimento de um *software*, mas também do desenvolvimento de novas ferramentas ou da melhoria das ferramentas que já existem para suporte e apoio ao *software*.

Então, chegamos à seguinte pergunta: ferramentas CASE são softwares que auxiliam a desenvolver novos softwares?



Sugerimos que você pare alguns minutos e tente formular sua própria resposta sobre isso.

Você pode perceber que o objetivo de qualquer indústria é satisfazer às necessidades de seus clientes entregando produtos com qualidade e aumentando, assim, a produtividade de seus processos de produção. Na indústria de *software*, esses objetivos não são diferentes.

A Engenharia de *Software* tem como objetivo o aperfeiçoamento da qualidade dos softwares desenvolvidos e o aumento da produtividade dos engenheiros que os desenvolvem, visando sistematizar sua manutenção, de modo que aconteça dentro de prazos estimados, com progresso controlado e usando métodos, tecnologias e processos em contínuo aprimoramento (REZENDE, 2005).

Passamos, no início da década de 1970 pela Crise do *Software*. Crise do *Software* é o termo que resume todos os problemas que permearam o desenvolvimento de *software* e que, de alguma forma, a Engenharia de *Software*, com seus processos e procedimentos estruturados tentou minimizar e, como muitos autores afirmam, colocou ordem numa atividade que aparentemente era caótica (atividade de desenvolver um *software*).

Um dos sintomas da crise – alguns dos quais permeiam até os dias atuais – é a dificuldade de suprir a demanda por novos softwares. As ferramentas CASE são promessas da Engenharia de Software para automatizar tarefas, diminuir o tempo de desenvolvimento e, assim, atender melhor à crescente demanda por novos softwares, sem contudo esquecer da qualidade necessária ao produto.

Em outras palavras, a preocupação em desenvolver ferramentas que automatizam o trabalho de engenheiros de software é uma das tentativas de aumentar a produtividade e a qualidade da indústria de software.

## Contexto histórico

Pressman (1995) lembra-nos do velho ditado sobre os filhos do sapateiro, ele passa a maioria do tempo fazendo sapatos para os outros e seus próprios filhos não têm sapatos feitos por ele.

Esse ditado é análogo ao ditado popular “em casa de ferreiro, o espeto é de pau”.

Segundo Pressman (1995), nos últimos 20 anos, a grande maioria dos engenheiros de softwares tem sido como os filhos do sapateiro: constroem softwares, sistemas complexos que automatizam o trabalho para os outros, mas, para si mesmos, não têm usado quase nada para automatizar o ambiente de trabalho. Recentemente, a Engenharia de Software era fundamentalmente uma atividade manual em que ferramentas eram usadas somente em últimos estágios do processo.

Por volta da década de 1950, engenheiros utilizavam régulas de cálculo e calculadoras mecânicas, livros e tabelas que continham as fórmulas e algoritmos que precisavam para efetuar a análise de um problema de engenharia. Tudo era feito manualmente, com qualidade, mas manualmente.

Passou uma década e esses engenheiros começaram a experimentar os benefícios da informática, principalmente do computador, na realização destas tarefas.

Já na década de 1970, todas as fórmulas que os engenheiros necessitavam estavam embutidas em um conjunto de programas de computador, usados para analisar uma ampla variedade de problemas, o que tornou inevitável o uso de tal ferramenta no dia a dia da profissão. Foi nesse contexto que a tecnologia começou a estreitar seus laços com o processo de manufatura, tendo os computadores como ferramenta oportunizadora do processo. Assim surgiu o primeiro elo entre o projeto auxiliado por computador – *Computer-Aided Design (CAD)* – e a manufatura auxiliada por computador – *Computer-Aided Manufacturing (CAM)* (PRESSMAN, 1995).

Com o passar dos anos e a institucionalização da produção de software como indústria, essas necessidades se intensificaram e, finalmente, os engenheiros de software ganharam sua primeira ferramenta auxiliada por computador, a Engenharia de Software Auxiliada por Computador – *Computer-Aided Software Engineering (CASE)*.

Atualmente, as ferramentas CASE fazem parte do dia a dia de trabalho do engenheiro de software. Por meio dela foi possível automatizar as atividades e melhorar a produtividade no desenvolvimento do produto de software (PRESSMAN, 2011).

A definição do que é produtividade na indústria de software ainda é controversa, pois a produtividade em software não é uma medida direta. Assim como nas outras engenharias, a

Engenharia de Software propõe algumas métricas para deter dados tangíveis sobre o processo e o produto de software, como, por exemplo, linhas de código – *Line-of-Code* (LOC) – e pontos por função – *Function-Point* (FP) –, que veremos adiante (PRESSMAN, 2011). Entretanto, há, no mínimo, cinco fatores que interferem na produtividade do software, são eles:

- » **Fatores humanos:** o tamanho e a experiência da organização de desenvolvimento.
- » **Fatores do problema:** a complexidade do problema a ser resolvido e o número de mudanças nos requisitos ou restrições de projeto.
- » **Fatores do processo:** técnicas de análise e projeto que são usadas, como: linguagens e ferramentas CASE disponíveis e técnicas de revisão.
- » **Fatores do produto:** confiabilidade e desempenho do sistema baseado em computador.
- » **Fatores relacionados ao recurso:** disponibilidade de ferramentas CASE, recursos de hardware e software.

Portanto, a disponibilidade de ferramentas CASE é considerada relevante, influenciando em até 40%, segundo Pressman (2011), na busca pelo aumento da produtividade do processo de software.

Para Sommerville (2005), as ferramentas de tecnologia CASE possuem facilidades gráficas para o planejamento, projeto e construção de sistemas. Elas podem ser utilizadas para gerar um esboço do programa, a partir de um projeto. Isso inclui código, implementação interfaces e, em vários casos, o desenvolvedor precisa apenas acrescentar pequenos detalhes da operação de cada componente do programa. Elas também podem incluir geradores de códigos, que, automaticamente, originarão código-fonte com base no modelo de sistema, e, também, algumas orientações de processo, que fornecem conselhos ao engenheiro de software sobre o que fazer em seguida.

## Classificação das CASE

De acordo com Sommerville (2005), em uma perspectiva de processo, isto é, quanto às fases do processo que a ferramenta automatiza, as CASE podem se dividir em três categorias:

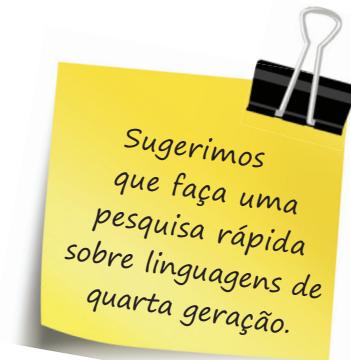
- **Front End ou Upper-CASE:** são aquelas ferramentas que dão apoio à análise e ao projeto, isto é, às fases iniciais do desenvolvimento do software.
- **Back End ou Lower-CASE:** são aquelas ferramentas destinadas a dar apoio à implementação e aos testes, como depuradores, sistemas de análise de programa, geradores de casos de testes e editores de programas.

- **I-CASE ou Integrated CASE:** são as ferramentas que têm como objetivo unir a Upper-CASE à Lower-CASE, isto é, cobrem todo o ciclo de vida do software.

Já Pressman (1995), em uma perspectiva de função, ou seja, de acordo com sua função específica que automatiza, as CASE podem ser classificadas em oito ferramentas. Observe a seguir a descrição de cada uma.

#### **Ferramenta de planejamento de sistemas comerciais:**

essa ferramenta tem como objetivo melhorar a compreensão de como a informação circula entre as várias unidades organizacionais. É considerado um “meta-modelo”, com base na qual sistemas de informação específicos são derivados. A informação comercial é considerada e modelada a partir da forma como circula pelas entidades comerciais não sendo considerada como requisito.



**Ferramenta de gerenciamento de projetos:** a maioria concentra-se em um elemento específico do gerenciamento ao invés de apoiar diversas atividades simultâneas. Ao ser usado um conjunto selecionado de ferramenta CASE, o gerente de projetos pode gerar estimativas de esforço, custo e duração, além de definir a estrutura de divisão de trabalho, planejar o cronograma e acompanhar os projetos continuamente. Além disso, essa ferramenta pode ser usada para compilar métricas e rastrear os requisitos.

**Ferramenta de apoio:** essa categoria, como o nome já diz, é de apoio. Suas funcionalidades são, dentre outras, apoiarativamente atividades de suporte, tais como a documentação de rede e de sistema, de garantia de qualidade, de gerenciamento de banco de dados e de configuração.

**Ferramentas de análise e projeto:** possibilitam que o engenheiro de software crie um modelo do sistema que será construído. Essas ferramentas também auxiliam na criação do modelo e na avaliação da qualidade do modelo.

**Ferramenta de programação:** são aquelas relacionadas aos compiladores, editores e depuradores que são necessárias nas atividades de desenvolvimento. Estão nessa categoria as linguagens denominadas de quarta geração, os geradores de aplicações e as linguagens de consulta a banco de dados.

**Ferramenta de integração e teste:** auxilia na aquisição de dados de testes, na análise do código-fonte, no planejamento, no gerenciamento e controle de testes.

**Ferramenta de prototipação:** dá suporte à criação de modelos para prototipação.

**Ferramenta de manutenção:** auxilia na execução de engenharia reversa, análise e reestruturação de código e na reengenharia.

Veja que há várias classificações para as ferramentas CASE, cada uma relacionada à funcionalidade específica (manutenção, prototipação, apoio etc.). Esta classificação pode ser feita tanto pela sua função quanto pelos papéis que desempenha como suporte aos desenvolvedores, gerentes e outros integrantes da equipe.

A Engenharia de Software Auxiliada por Computador pode ser tão simples quanto uma única ferramenta que suporte uma atividade de Engenharia de Software específica ou tão complexa quanto um ambiente completo que abrange ferramentas, banco de dados, pessoas, hardware, rede, sistemas operacionais, padrões e uma infinidade de outros componentes (PRESSMAN, 2011). Enfim, todas as ferramentas “softwares” que, de alguma forma, auxiliam nos trabalhos de um engenheiro de software, podem ser consideradas como CASE.

Há várias classificações de ferramentas CASE que apoiam as diversas fases do processo de software. Para cada uma dessas classificações, há, no mercado, inúmeras ferramentas disponíveis. Entre elas, podemos citar: *Poseidon, Rational, ErWin, Oracle Designer, Genexus, Clarify, Dr.CASE, MultiCASE, Paradigm, PowerDesigner, Together, Cognos, CoolGen, Smart, Theseus, BPWin, Arena, Visio, Brio, Microstrategy*.



## Informação

O SEI (*Software Engineering Institute – Instituto de Engenharia de Software*), *Carnegie Mellon University, Pittsburgh, Pennsylvania*, Estados Unidos, desenvolveu um processo de adoção de ferramentas CASE. O modelo tem como postulados seis estágios para um processo de adoção de ferramentas CASE.



## Explore

Mais informações sobre o modelo SEI podem ser encontradas no site do Instituto Nacional de Pesquisas Espaciais. Disponível em: [www2.dem.inpe.br/ijar/GuiaCASE.doc](http://www2.dem.inpe.br/ijar/GuiaCASE.doc). Acesso em: 5 jul. 2010.

Idealmente, as CASE têm como promessas:

- encorajar um ambiente interativo;
- reduzir custos de manutenção;
- melhorar a qualidade do produto de software;
- agilizar o processo de desenvolvimento;
- aumentar a produtividade.

Com os seguintes benefícios decorrentes da utilização de ferramentas CASE:

- CASE de gerenciamento de configuração e documentação são, geralmente, mais aceitas como mecanismo de melhoria do software;
- benefícios controversos de CASE de análise e projeto, engenharia reversa e ferramentas de geração de códigos disponíveis comercialmente;

- ganhos variando de 10% a 30% resultante do uso de CASE na análise e projeto;
- ganhos verdadeiros ocorrem somente depois de um ou dois anos de experiência;
- ganhos variáveis de produtividade;
- modestos ganhos de qualidade;
- documentação melhorada (aumento da manutenibilidade);
- melhoria na comunicação;
- imposição de metodologia e padrões.

Como os benefícios das CASE são ainda relativos, as empresas que se dispuserem a adquirir uma ferramenta para automatizar o processo de *software*, devem considerar alguns itens da CASE em questão, como, por exemplo:

- custo (investimentos) para adotar a tecnologia CASE;
- consistência entre os processos e métodos suportados pelas ferramentas CASE e os processos e métodos utilizados na organização;
- mecanismos de suporte necessários para ferramentas CASE (por parte do fornecedor);
- limites da ferramenta quanto ao tamanho do projeto;
- complexidade da adoção e usabilidade mínima. Complexidade dos processos de adoção das ferramentas CASE;
- capacidade de acomodar mudanças, uma vez que os requisitos se modificam;
- permissão da engenharia reversa dos softwares desenvolvidos sem usar uma ferramenta CASE.

O sucesso ou falha do esforço de adoção da CASE depende muito da habilidade de uma organização para gerenciar custos de curto e de longo prazo.

Portanto, devem ser consideradas algumas implicações de curto prazo:

- um potencial decaimento na produtividade;
- insatisfação de parte dos funcionários ao adotar a nova tecnologia;
- mudanças nos processos e métodos;
- treinamento potencialmente extensivo;
- custos significativos.

Apesar de serem consideradas, muitas vezes, ferramentas de custo elevado e de utilização complexa, uma vez inserida e institucionalizada no processo de desenvolvimento de *software*, certamente, trarão benefícios ao projeto por meio do aumento da produtividade.

Assim, a Engenharia de Software oferece métodos e técnicas para desenvolver ferramentas automatizadas para auxiliar no trabalho de profissionais das mais diversas áreas de atuação. Tais ferramentas – softwares – são bem aceitas, prova disso é a crescente busca por novos e cada vez mais complexos sistemas.

## Qualidade de software



Vimos no início deste Material Teórico que as ferramentas CASE auxiliam no processo de desenvolvimento de produtos de *software* com qualidade, entretanto, durante muito tempo, as questões de qualidade eram subjetivas e, aos poucos, esse ambiente foi sendo alterado.

Quando falamos em certificação de qualidade em empresas, geralmente pensamos em certificações ISO, especificamente as relacionadas à ISO 9000 e suas derivações.

Certificações ISO possuem restrições, listas de atributos e níveis que as empresas devem obter relacionadas à qualidade para que possam ser certificadas e tudo parte do princípio da definição do que é qualidade dentro do contexto a ser avaliado.

Segundo a **norma ISO 9000**, versão 2000, a qualidade é o grau em que um conjunto de características inerentes a um produto, processo ou sistema cumpre os requisitos inicialmente estipulados para estes (WIKIPEDIA, 2007a).

Com relação aos produtos de *software* isso não será diferente. Precisamos verificar o que é qualidade nesse contexto para entendermos porque empresas “correm tanto” atrás de um *software* com qualidade.

Para Pressman (2011), Qualidade de *Software*, que é uma área que pertence à Engenharia de *Software*, objetiva atingir e garantir a qualidade final do produto, por meio das definições e normatizações dos processos de desenvolvimento.

Apesar dos diversos modelos aplicados na questão “Qualidade de *Software*” atuarem durante todo o processo de desenvolvimento, o foco principal está na satisfação do cliente ao receber o produto pronto. É importante que se procure garantir que o sistema cumpra com todas as especificações acordadas anteriormente entre a empresa desenvolvedora e o cliente. É nesse momento que percebemos a importância de uma definição de requisitos bem elaborada.

Entretanto, é uma visão simplista dizer que avaliação de qualidade de um *software* só pode ser feita depois que o *software* foi entregue ao cliente. Deve-se garantir a qualidade desde o início da construção do *software*, pois controlamos a sua fabricação passo a passo e medimos a sua qualidade antes que ele saia da fábrica.

Você provavelmente já ouviu falar que a qualidade das partes garante a qualidade do todo; é mais ou menos isso que acontece: ao garantirmos a qualidade das partes do *software*, estamos garantindo a qualidade total do produto.

Existem fatores internos e externos que estão relacionados à qualidade.

- Fatores de qualidade externa são aqueles que podem ser percebidos por pessoas fora da equipe de desenvolvimento – cliente ou eventuais usuários. A partir da observação de fatores específicos, o cliente pode perceber a qualidade ou não do produto de *software*. Enquadram-se nesta classe de fatores de qualidade externa: desempenho, facilidade de uso, correção, confiabilidade, extensibilidade, dentre outros.
- Fatores de qualidade interna são aqueles que estão mais relacionados à visão de um programador, particularmente aquele que vai assumir as tarefas de manutenção do *software*. Enquadram-se nesta classe de fatores de qualidade interna: modularidade, legibilidade, portabilidade, manutenibilidade, dentre outros.

Mesmo observando que são diferentes entre si, é a garantia da qualidade de fatores externos e internos que nos assegura um bom produto de *software*.

## Fatores de qualidade de *software*

Os fatores que afetam a qualidade de *software* podem ser categorizados em:

- Fatores que podem ser medidos diretamente (erros de execução).
- Fatores que podem ser medidos apenas indiretamente (usabilidade do *software*).

Vejamos alguns fatores que devem ser considerados (PRESSMAN, 2006, 2011). Para cada um dos fatores apresentados, algumas perguntas podem ser realizadas para perceber a existência ou não deles:

### **Corretitude (Ele faz aquilo que eu quero?)**

É a capacidade dos produtos de *software* de executarem suas funções precisamente, conforme definido nos requisitos e na especificação.

### **Confiabilidade (Ele se comporta com precisão o tempo todo?)**

É a capacidade de o sistema funcionar mesmo em condições anormais. É um fator diferente da corretitude, pois um sistema pode ser correto sem ser confiável, ou seja, ele funciona, mas não o tempo todo e em todas as condições.

### **Flexibilidade (Posso mudá-lo?)**

É a facilidade com a qual podem ser introduzidas modificações nos produtos de *software*. Todo *software* é considerado, em princípio, “flexível” e, portanto, passível de modificações. No entanto, esse fator nem sempre é muito bem entendido, principalmente em se tratando de pequenos programas.

Por outro lado, para softwares de grande porte, esse fator atinge uma importância considerável e pode ser alcançado a partir de dois critérios importantes:

- A **simplicidade de projeto**, ou seja, quanto mais simples e clara a arquitetura do *software*, mais facilmente as modificações poderão ser realizadas.
- A **descentralização**, que implica na maior autonomia dos diferentes componentes de *software*, de modo que a modificação ou a retirada de um componente não implique uma reação em cadeia que altere todo o comportamento do sistema, podendo inclusive introduzir erros antes inexistentes.

### **Reusabilidade (Serei capaz de reutilizar parte do software?)**

É a capacidade de os produtos de *software* serem reutilizados, totalmente ou em parte, para novas aplicações. Esse é um conceito fundamental nos dias de hoje.

Essa característica de reusabilidade é uma necessidade e vem da observação de que alguns dos componentes de *software* obedecem a um padrão comum de execução, restrições e requisitos, o que permite que essas similaridades possam ser exploradas e incorporadas para solucionar outras classes de problemas.

Esse fator permite, principalmente, diminuir o tempo de desenvolvimento de um *software*, gerando economia e qualidade maiores, ou seja, ao utilizarmos objetos (programas ou parte deles) já desenvolvidos, menos algoritmos precisam ser escritos, o que significa menos esforço e menor risco de ocorrência de erros.

#### **Compatibilidade (Serei capaz de compor uma interface com outro sistema?)**

A compatibilidade corresponde à facilidade com a qual produtos de *software* podem ser combinados com outros. Esse é um fator relativamente importante, dado que um produto de *software* é construído (e adquirido) para trabalhar convivendo com outros *softwares*.

#### **Eficiência (Ele rodará em meu hardware tão bem quanto possível?)**

A eficiência está relacionada com a utilização racional dos recursos de hardware e de sistema operacional da plataforma onde o *software* será instalado.

#### **Portabilidade (Serei capaz de utilizá-lo em outra máquina?)**

A portabilidade consiste na capacidade de um *software* em ser instalado para diversos ambientes operacionais e de hardware. Por termos várias plataformas de processadores e sistemas operacionais, essa não é uma característica fácil de ser atingida.

#### **Usabilidade (Ele foi projetado para o usuário?)**

Esse fator é certamente um dos mais fortemente detectados pelos usuários do *software*, ou seja, é medido por meio da facilidade de se utilizar o produto.

#### **Manutenibilidade (Posso consertá-lo?)**

Esse fator é relacionado ao esforço exigido para localizar e reparar erros num programa, além de adequá-lo a novas versões e atualizá-lo de forma eficaz e eficiente.

#### **Testabilidade**

Relacionado ao esforço despendido para testar um *software* a fim de garantir que execute todas as funções para qual foi projetado.

#### **Integridade (Ele é seguro?)**

Se o sistema pode ser facilmente acessado por pessoas não autorizadas.

Esses itens citados podem, de acordo com referências existentes, ser medidos e, por meio do resultado dessa medição, ser definido seu fator de qualidade.

## **Garantia de qualidade de software**

A garantia da qualidade é uma atividade fundamental para qualquer negócio que gere produtos ou serviços.

A garantia de qualidade de *software* engloba algumas atividades como: teste, padronizações e procedimentos formais que são aplicados ao processo de engenharia de *software* para o desenvolvimento de *software* com qualidade, controle de mudança, medição e manutenção.

Algumas dessas atividades veremos a seguir, outras, sugerimos que faça uma pesquisa, porque é de fundamental importância conhecê-las, mas devido à nossa carga horária da disciplina, restrições de conteúdo tiveram que ser realizadas.

## Métrica de qualidade de software

A possibilidade de estabelecer uma medida da qualidade é um aspecto importante para a garantia de um produto de *software* com algumas das características definidas anteriormente.

Mas como medir, por exemplo, o quanto um *software* será fácil ou não de dar manutenção, ou será seguro?

É nesse contexto de “como medir” que um novo conceito na área de ES é inserido: conceito de métrica.

Uma vez que as medidas quantitativas (mensuráveis) têm-se provado eficientes em vários ramos da ciência, cientistas de computação têm trabalhado arduamente para aplicar métodos similares no desenvolvimento de *software*.

Segundo Wikipédia (2007d, 2007e), a **métrica de software** é a medida de alguma propriedade do *software* ou da sua especificação. A métrica é utilizada para calcular orçamentos, desempenho dos programadores etc.

Para Cavano e MacCall (*apud* PRESSMAN, 1996, p. 753):

A determinação da qualidade é fundamental nos eventos cotidianos – concursos de degustação de vinhos, eventos esportivos, concursos de talento, etc. Nessas situações, a qualidade é julgada na maneira mais fundamental e direta: uma comparação lado a lado dos objetos sob condições idênticas e com conceitos predeterminados. O vinho pode ser julgado de acordo com a clareza, cor, buquê, sabor, etc. Porém, esse tipo de julgamento é muito subjetivo; para ter qualquer valor absoluto, ele deve ser feito por um especialista.

A subjetividade e a especialização também se aplicam na determinação da qualidade de *software*. Para ajudar a resolver esse problema, uma definição mais precisa de qualidade de *software* é necessária, bem como uma forma de derivar medições quantitativas de qualidade de *software* para análise objetiva [...]. Uma vez que não existe essa coisa de conhecimento absoluto, ninguém deve esperar medir qualidade de *software* exatamente, porque cada medição é parcialmente imperfeita. Jacob Bronowsky descreveu esse paradoxo do conhecimento desta maneira: “Ano a ano deparamo-nos com instrumentos cada vez mais precisos com os quais podemos observar a natureza com mais precisão. E quando olhamos para as observações ficamos desconsertados ao ver que elas ainda são vagas, e achamos que elas continuam tão incertas como sempre”.

Por essa citação, percebemos que medir qualidade não é coisa fácil, mas nem por isso especialistas deixaram de tentar criar modelos para que chegássemos o mais próximo possível de uma medição eficaz.



## Atenção

É importante salientar que não medimos diretamente a qualidade de *software*, mas a manifestação dessa qualidade durante sua execução.

Nesse momento, não teremos como abordar com profundidade as técnicas de medição, mas citaremos algumas para que você dedique algum tempo para pesquisar sobre estas teorias/técnicas de medição.

## Métricas de dimensão e complexidade

Têm sido propostas inúmeras métricas para medir a dimensão e complexidade de um programa. Elas são apresentadas em conjunto pois, na maior parte das vezes, a mesma métrica é apresentada quantificando ora a dimensão, ora a complexidade.

### A- Linhas de Código Fonte – Lines of Code (LOC)

É uma métrica de dimensão que, apesar de criticada, é ainda a mais utilizada.

Embora aparentemente simples, obriga a uma definição inequívoca das regras de contagem de linhas, nomeadamente no tocante ao tratamento a dar às linhas em branco e de comentário, instruções não executáveis, diretivas de compilação, múltiplas instruções por linha ou múltiplas linhas por instrução, bem como no caso de reutilização de código.

### B- Métricas de Halstead

É um conjunto de métricas proposto por Maurice Halstead que se baseia na teoria da informação e que o autor designou por “Software Science”.

Ele usa medidas primitivas para desenvolver expressões para o comprimento global do programa, o volume mínimo potencial para um algoritmo, o volume real (medido em bits), o nível do programa e outras características, como o esforço do desenvolvimento, o tempo de desenvolvimento e até mesmo o número projetado de falhas no *software*.

São utilizadas bases de cálculos matemáticos para se chegar a um valor métrico da qualidade e complexidade do *software*.

Pressman (2011) defende que a métrica de comprimento de Halstead é objetiva e melhor que a LOC.

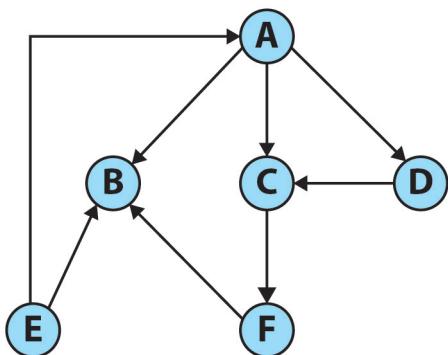
### C- Métricas de McCabe

A primeira e mais conhecida métrica proposta por Thomas McCabe é a *métrica de complexidade ciclomática*.

Ela pressupõe que a complexidade depende do número de decisões (complexidade ciclomática), é adimensional e corresponde ao número máximo de percursos linearmente independentes através de um programa.

Os caminhos podem ser representados através de um gráfico orientado em que os nós sinalizam uma ou mais instruções sequenciais e os arcos orientados indicam o sentido do fluxo de controle entre várias instruções (Figura 1).

**Figura 1:** Complexidade do gráfico de fluxo de controle.



Outras métricas que valem a pena pesquisar e conhecer:

- **Métrica dos Nós:** proposta por Woodward.
- **Métrica dos Fluxos de Informação:** proposta por Henry e Kafura.
- **Métrica voltada para Orientação Objeto.**

## Estimativa de software

Quando falamos em estimativa de *software*, uma coisa deve ficar clara: é difícil estabelecer se é possível desenvolver o produto desejado pelo cliente antes de conhecer os detalhes do projeto.

Por isso que uma boa definição de requisitos e os encontros periódicos com os clientes são fundamentais para estimar o tempo de desenvolvimento, custo, tamanho do projeto etc.

O desenvolvimento de um *software* é um processo gradual de refinamento e devemos sempre lembrar que:

- A incerteza da natureza do produto contribui para a incerteza da estimativa.
- Requisitos e escopo mudam.
- Defeitos geralmente são encontrados e demandam retrabalho.
- A produtividade varia de pessoa para pessoa.

O **processo de estimativa** envolve 5 etapas básicas:

- Estimar o Tamanho do Produto.
- Estimar o Tempo.
- Estimar o Esforço.
- Estimar o Custo (envolve 4 fatores).
- Estimar o Prazo.

É viável fornecer estimativas dentro de uma faixa permitida e, com o passar do tempo, a partir do momento em que se conhece mais e mais o projeto, refinar essa faixa.

## **Estimativa de Tamanho**

- É a dimensão do *software* a ser produzido. Seu tamanho e quantidade, por exemplo:
  - N° linhas de código, n° pontos de função, n° de requisitos, pontos de casos de uso etc.

## **Estimativa de Tempo**

- Após desenvolver uma estimativa do volume de trabalho a ser feito, não é fácil estimar o período em que o projeto será executado. Para que não estimemos outros fatores como custo de forma inadequada, a relação entre tempo e pessoa deve ser bem dimensionada/estimada.

## **Estimativa de Esforço**

- É a técnica mais comum para apurar os custos de qualquer projeto de desenvolvimento.
- A estimativa de esforço tem início com a definição do escopo do projeto e as funções que deverão conter.
- O planejador estima o esforço (por exemplo, pessoas/mês), que seria exigido para conclusão de cada tarefa de Engenharia de *Software*, para cada função de *software*. Taxas de mão de obra (isto é, custo/esforço unitário) são aplicadas em cada uma das tarefas de Engenharia de *Software*.

## **Estimativa de Custo**

- Aqui o objetivo é calcular, antecipadamente, todos os custos associados ao sistema: construção, instalação, operação e manutenção.

## **O Custo da Construção**

- Uma vez que o custo está associado ao número de pessoas envolvidas no desenvolvimento do sistema – tais como burocratas, diretores, membros da comunidade usuária, consultores e programadores, membros da auditoria, do controle de qualidade ou da equipe de operações –, ele deve ser cuidadosamente mensurado.

## **O Custo da Instalação do Sistema**

- Esse custo está relacionado ao modo como o cliente terá acesso ao produto. Se ele for instalar sozinho o sistema, é um valor, se for um sistema de grande porte e que precisa de uma equipe de instalação e treinamento, é outro. Quanto maior o sistema, maiores os custos, pois teremos que prever: custo de treinamento do usuário, custo de conversão de banco de dados, custo de instalação do fornecedor, custo da aprovação legal etc.

## O Custo Operacional

- Entra em ação após a instalação do sistema. Haverá um custo para o usuário manter a operação do produto. Nesse custo, deve ser previsto como e quando o cliente, com o novo produto, poderá economizar dinheiro, a partir da utilização desse novo instalado. Os custos operacionais mais comuns são: custos de hardware e suprimentos, custos de software, custo de pessoal, custo de manutenção etc.

## O Custo de Manutenção ou Falhas

- Por não termos sistemas perfeitos, esse custo deve ser bem dimensionado. O preço (direto ou indireto) a ser pago por todos se um sistema crucial ficar horas ou dias sem operação pode ser incomensurável e gerar inúmeros incômodos no momento em que isso acontece.

## Estimativas de Prazo

- Geralmente são dirigidas a datas fornecidas pelo cliente e devem, sempre que possível, serem respeitadas.

## Fator Humano

- Quando os objetivos para o desenvolvimento de sistemas não são claros, as pessoas passam a deduzir e criar o produto a partir do que acreditam que seja necessário, desenvolvendo, em inúmeros casos, sistemas inadequados e, consequentemente, métricas falhas, gerando uma expectativa negativa entre o cliente e os técnicos responsáveis, isto é, uma estimativa irreal.
- As pessoas são sensíveis aos estímulos externos e por eles são influenciadas. Um analista, ou um grupo de analistas, disposto a estimar o tempo e custo de um projeto não poderia deixar de dar a devida relevância a esse fato.

## Engenharia Humana

- Para tentar amenizar a dificuldade e estabelecer critério para a estimativa em relação às pessoas, surge o conceito de Engenharia Humana, que consiste em aplicar conceitos de psicologia para projetar uma interação homem-computador de alta qualidade. Do ponto de vista do especialista em Engenharia Humana ou interface homem-computador, o homem é tratado como elo de coleta e processamento de dados.

Assim, podemos concluir que as estimativas jamais poderão ser precisas e exatas, pois não são compostas apenas por fatores técnicos, “contáveis” e palpáveis que fazem parte de um projeto, mas também por fatores humanos integrados (sentimentos, políticas, crenças, percepção, experiência etc.), assim como o ambiente e outras características mais, que não podemos estimar de forma absoluta. Entretanto, devem ser analisadas por meio dos embasamentos teóricos existentes sobre o tema. Afinal, estimar não é adivinhar e estimativas mal dimensionadas geram problemas.

## Material Complementar



### Explore

O objetivo do material complementar é lhe ajudar a entender, sob uma ótica diferente daquela da professora conteudista, assuntos abordados nas unidades teóricas.

É fundamental a leitura deste material para o melhor entendimento sobre o assunto.

Como nesta unidade abordamos os conceitos gerais da Engenharia de Software, nossa sugestão de material complementar é o capítulo 27, intitulado Gerenciamento de qualidade, no seguinte livro:

SOMMERVILLE, I. **Engenharia de Software**. 8. ed. São Paulo: Pearson, 2007. p. 423-438.

## Referências

### Bibliografia fundamental

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

### Bibliografia básica

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação**. 4. ed. Rio de Janeiro: LTC, 1999.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação Gerenciais**. Administrando a empresa digital. 5. ed. São Paulo: Pearson Education do Brasil, 2006.

PFLEEGER, S. L. **Engenharia de Software**: teoria e prática. São Paulo: Prentice Hall, 2004.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 2006.

SOMMERVILLE, I. **Engenharia de Software**. 6. ed. São Paulo: Pearson Addison Wesley, 2005.

### Bibliografia complementar

ALCADE, E.; GARCIA, M.; PENUELAS, S. **Informática Básica**. São Paulo: Makron Books, 1991.

FAIRLEY, R. E. **Software engineering concepts**. New York: McGraw-Hill, 1987.

IEEE. **Software Engineering Standards**. 2013. Disponível em: <[http://www.ieee.org/portal/inno\\_vate/products/standard/ieee\\_soft\\_eng.html](http://www.ieee.org/portal/inno_vate/products/standard/ieee_soft_eng.html)>. Acesso em: 10 dez. 2013.

LUKOSEVICIUS, A. P.; CAMPOS FILHO, A. N.; COSTA, H. G. **Maturidade em gerenciamento de projetos e desempenho dos projetos**. Disponível em: <[www.producao.uff.br/conteudo/rpep/.../RelPesq\\_V7\\_2007\\_07.doc](http://www.producao.uff.br/conteudo/rpep/.../RelPesq_V7_2007_07.doc)>. Acesso em: 12 nov. 2013.

MAFFEO, B. **Engenharia de software e especialização de sistemas**. Rio de Janeiro: Campus, 1992.

MICHAELIS. **Moderno dicionário da língua portuguesa**. São Paulo: Cia. Melhoramentos, 1998.

PARREIRA JÚNIOR, W. M. **Apostila de Engenharia de Software**. Disponível em: <[http://www.waltenomartins.com.br/ap\\_es\\_v1.pdf](http://www.waltenomartins.com.br/ap_es_v1.pdf)>. Acesso em: 13 nov. 2013.

PAULA FILHO, W. P. **Engenharia de Software**: fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: LTC, 2001.

**Revista Engenharia de Software.** Disponível em: <<http://www.devmedia.com.br/revista-engenharia-de-software-magazine>>. Acesso em: 12 nov. 2013.

VONSTA, A. **Engenharia de Programas**. Rio de Janeiro: LTC, 1983.

WIENNER, R.; SINCOVEC, R. **Software engineering with Modula 2 and ADA**. New York: Wiley, 1984.

WIKIPEDIA (2007a). **ISO 9000**. Disponível em: <[http://pt.wikipedia.org/wiki/ISO\\_9000](http://pt.wikipedia.org/wiki/ISO_9000)>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007b). **Melhoria de processo de software brasileiro**. Disponível em: <<http://pt.wikipedia.org/wiki/MPS.BR>>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007c). **CMMI**. Disponível em: <<http://pt.wikipedia.org/wiki/Cmmi>>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007d). **Engenharia de Software**. Disponível em: <[http://pt.wikipedia.org/wiki/Engenharia\\_de\\_software](http://pt.wikipedia.org/wiki/Engenharia_de_software)>. Acesso em: 01 fev. 2007.

# Anotações







**Educação a Distância**  
Cruzeiro do Sul Educacional  
*Campus Virtual*

[www.cruzeirodosulvirtual.com.br](http://www.cruzeirodosulvirtual.com.br)  
Campus Liberdade  
Rua Galvão Bueno, 868  
CEP 01506-000  
São Paulo SP Brasil  
Tel: (55 11) 3385-3000

