Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

ÍTALO MAGNO PEREIRA

BENEFÍCIOS DA MODULARIZAÇÃO NA OBTENÇÃO DE SOFTWARE DE QUALIDADE

Belo Horizonte
2011

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

BENEFÍCIOS DA MODULARIZAÇÃO NA OBTENÇÃO DE SOFTWARE DE QUALIDADE

por

ÍTALO MAGNO PEREIRA

Monografia de Final de Curso

Mariza Andrade da Silva Bigonha
Orientadora

Belo Horizonte
2011

ÍTALO MAGNO PEREIRA

BENEFÍCIOS DA MODULARIZAÇÃO NA OBTENÇÃO DE SOFTWARE DE QUALIDADE

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração: Engenharia de Sofrware

Orientadora: Prof.ª Dra. Mariza Andrade da Silva Bigonha

Belo Horizonte
2011

A minha família pelo apoio inestimável.

Aos professores desta instituição pelo aprendizado que obtive.

Aos colegas de curso pelo companheirismo e amizade.

AGRADECIMENTOS

Primeiramente a Deus por todo o cuidado com a criação.

Aos meus pais e irmãos pelo apoio nesta difícil jornada, especialmente a minha mãe.

Aos professores pela ótima qualidade do curso.

Aos colegas de curso pelo companheirismo e pela troca de experiências.

"A qualidade nunca se obtém por acaso;
ela é sempre o resultado do esforço inteligente."
John Ruskin

RESUMO

O uso de sistemas informatizados é cada vez mais comum em diversas áreas, seja financeira, educacional, entretenimento ou outras. O bom funcionamento destes sistemas pode ser considerado um fator importante para o sucesso de negócios e pode ser crítico para a segurança humana, como por exemplo, em sistemas embarcados de aeronaves. A falha destes sistemas pode ser desastrosa causando grandes prejuízos e em alguns casos fatalidades. Logo a qualidade de software não deve ser tratada como tema secundário, deve-se utilizar técnicas reconhecidas para o desenvolvimento dos produtos de software que lhe garantam boa qualidade. O objetivo principal desta monografia é apresentar o resultado de um estudo sobre o impacto da programação modular, seus conceitos e princípios na produção de software de boa qualidade. O uso adequado da programação modular e de seus princípios permitem desenvolver software de boa qualidade, com um bom nível de reusabilidade, e consequentemente, com custos menores de manutenção.

Palavras-chave: modularidade, qualidade, manutenibilidade, reusabilidade

ABSTRACT

The use of computerized systems becomes more common in several fields, including financial, educational, entertainment and others. The good operation of these systems can be considered a decisive factor for the success of business and can be critical for the human security, for instance, in systems used in airplanes. The software quality must not be treated as a secondary issue. So, to assure good quality of the software products the designers must use recognized techniques in their development. The main aim of this work is to present the results of a study done about modular programming, their concepts and principles. The modular programming and its principles help to reach good quality products, with high maintainability and a good level of reusability, and, as consequence, with lowest costs.

Keywords*:* modularity, quality, maintainability, reusability

LISTA DE FIGURAS

LISTA DE SIGLAS

OO	Orientado(a) por Objetos
POO	Programação Orientada por Objetos
TAD	Tipo Abstrato de Dados
NBR	Norma Brasileira
ISO	International Organization for Standardization

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

Durante muito tempo busca-se o desenvolvimento de software de qualidade (Staa, 2000), (Meyer, 1997), (Myers, 1975) entre outros. Ferreira (2006) cita que nas últimas décadas muito se estudou sobre os processos de desenvolvimento de software com o objetivo de desenvolver software com melhor qualidade e custos menores. Ferreira destaca como um dos fatores de maior importância em relação à qualidade do software a manutenibilidade, que significa o quão fácil é realizar alterações ou implementar novas funcionalidades. A manutenibilidade é o principal responsável pelo custo do produto, sendo muitas vezes superior ao custo da produção do software e pode ser considerado como um fator determinante para sua viabilidade. A dificuldade de manutenção pode ser explicada pelos processos de desenvolvimento utilizados que geram produtos pouco flexíveis e de baixa estabilidade, e muitas vezes, sistemas muito acoplados.

A abordagem orientada por objetos tem se mostrado facilitadora na construção de software de boa qualidade, isto porque este paradigma permite que se construa software de forma modular. Ou seja, software contendo uma estrutura flexível e estável, com partes independentes, com baixo acoplamento. Entretanto, cuidados devem ser tomados, pois apenas usar o paradigma orientado por objetos não é o suficiente para se obter os benefícios mencionados. Para obtê-los deve-se utilizar boas práticas para o desenvolvimento de sistemas, estas são resultados de estudos e observações, tais como os critérios, regras e princípios de programação modular (Meyer, 1997).

Segundo Santos (2006) a abordagem orientada por objetos mostra-se mais adequada ao desenvolvimento de sistemas maiores ou de maior complexidade, por resolver problemas que uma das abordagens anteriores, a abordagem estruturada, não resolvia. A abordagem estruturada dá ênfase as funções e a abordagem orientada por objetos dá ênfase aos objetos, encapsulando em uma única estrutura os atributos e as operações que acessam ou alteram estes atributos. Com esta atribuição de responsabilidade o sistema é desenvolvido via módulos mais coesos, diminuindo a complexidade dos sistemas e minimizando o impacto de suas manutenções. Outro

benefício citado é obtenção de maior reusabilidade, devido ao sistema ter sido desenvolvido por meio de classes prestadoras de serviços e não mais uma seqüência de funções, o reúso é beneficiado pela utilização destes serviços.

Na orientação por objetos há uma série de conceitos importantes para seu entendimento, Oliveira et al. (2006) destacam alguns: classe, objeto, encapsulamento, herança e polimorfismo. Classe é a descrição de um grupo de objetos, que possuem propriedades (atributos), comportamento (operações), e pode ser definida como um gabarito para criar objetos. Objeto é a instância de uma classe em tempo de execução. Cada objeto tem seus próprios dados encapsulado, estado e comportamento. Encapsulamento é a técnica utilizada para ocultação das informações do objeto, permitindo que os mesmos possam ser acessados somente mediante utilização das operações disponíveis no objeto. Herança é o mecanismo utilizado para possibilitar o reúso de atributos e operações de um objeto. Polimorfismo é a qualidade de um objeto de ser capaz de assumir diferentes formas.

O objetivo desta monografia é apresentar o resultado de um estudo sobre o impacto da modularidade no processo de desenvolvimento de software de qualidade. Para obter o resultado descrito nesse trabalho estudamos:

- os conceitos relacionados com a programação modular e seus princípios;
- os recursos oferecidos pela orientação por objetos; e,
- os fatores de qualidade de software.

1.1. Estrutura da Monografia

Esta monografia está organizada da seguinte forma: Seção 2 descreve sobre a programação modular, destacando os conceitos mais importantes para atender os objetivos pretendidos neste trabalho, por exemplo, dentre outros, como um módulo deve ser estruturado, o que é coesão, acoplamento e como a presença dessas duas métricas influem na manutenção de um software. Seção 3 mostra o que pode ser considerado como software de boa qualidade de acordo com o estado da arte,

destacando a opinião de vários especialistas. Seção 4 apresenta os benefícios da modularidade na produção de software de boa qualidade e conclui este trabalho.

2. PROGRAMAÇÃO MODULAR

Esta seção tem como objetivo apresentar os principais conceitos relacionados com a programação modular. Descrever coesão e acoplamento, aspectos importantes considerados para avaliar se um processo de desenvolvimento é considerado modular. E não menos importante, apresentar brevemente os benefícios e dificuldades para o desenvolvimento de software modular.

A programação modular é tida como base para o desenvolvimento de programas de forma organizada. O particionamento de programas em módulos reduz a complexidade do programa e permite o trabalho em equipe, facilita a gerência de desenvolvimento, aumenta o controle e a garantia da qualidade, possibilita o reúso de módulos já existentes e possibilita a redução do tempo necessário para desenvolvimento do programa [Staa 2000].

O desejo de reutilizar componentes já existentes quando se desenvolve um novo sistema é um dos objetivos que podem ser alcançados via da programação modular. Este objetivo é atingido com o desenvolvimento de módulos que podem ser compostos com uma variedade de outros módulos. A reutilização de módulos já escritos possui muitas vantagens, dentre elas destacam-se: redução do esforço para implementação de um novo programa e melhoria da qualidade com base no uso de componentes homologados e maduros (Staa, 2000).

2.1. Definições de Programação Modular

Segundo Meyer (1997), o módulo é a menor unidade na qual um sistema pode ser decomposto. Para Staa (2000), o módulo é um artefato de programação que pode ser desenvolvido e compilado separadamente dos demais artefatos que compõe determinado problema. São exemplos de módulos: classes ou tipos abstratos de dados (TAD's). Devido a esta independência entre módulos, a programação modular

possibilita a criação de software flexível e de boa qualidade por construção, sendo a qualidade uma conseqüência do processo de desenvolvimento.

Componente é definido por Staa (2000) como um conjunto de um ou mais módulos, responsáveis pela formação de um todo coerente e cuja funcionalidade é bem definida. Afirma ainda que os componentes devem ser integrados ao programa sem que sofram alterações.

Todos os módulos oferecem serviços, estes serviços são oferecidos por suas interfaces, Staa (2000) define interface como o meio pelo qual os módulos trocam informações.

De acordo com Ferreira (2006), modularidade é a característica de construção de software por meio de unidades básicas, também chamadas de módulos, e pode ser vista como o principal aspecto na construção de software flexível. Um bom projeto de software possui módulos independentes, possibilita que estes sejam entendidos, implementados e alterados separadamente, sem a necessidade do conhecimento prévio de outros módulos do sistema. Se isso não for possível, a sua alteração deve causar o mínimo de impacto possível sobre os demais módulos. Essas características estão relacionadas com dois conceitos muito importantes, a coesão e o acoplamento, descritos, em mais detalhes, respectivamente nas duas subseções a seguir.

Para Myers (1975) em um projeto modular ótimo os elementos externos ao módulo devem ter seu relacionamento minimizado o máximo possível. Myers exibe duas maneiras de alcançar modularidade. A primeira diz respeito a minimizar o relacionamento entre módulos, denominado acoplamento entre módulos. A segunda diz respeito a maximizar o relacionamento entre os elementos internos de um módulo. Ela é chamada de coesão interna de módulos.

2.2. Coesão

De acordo com Sierra e Bates (2006) o termo coesão pode ser definido como o grau em que uma classe possui um único e bem definido propósito. A coesão indica o quanto estão relacionados os procedimentos e atributos internos de um módulo. O principal benefício de classes (módulos) altamente coesas é a sua facilidade de manutenção. Outro benefício importante é que elas se tornam mais reutilizáveis do que outras. Todo módulo deveria ser construído objetivando a alta coesão.

Myers (1975) classificou a coesão considerando do pior para o melhor caso. Sua classificação é apresentada a seguir.

1. Coesão coincidental. Esse tipo de coesão ocorre quando não há significado aparente entre os elementos internos de um módulo. Não é possível descrever a função do módulo, pois seu conteúdo parece ter sido reunido pelo acaso. É considerado o pior tipo de coesão por ser prejudicial a todos os fatores internos de qualidade de software, a saber, modularidade, manutenção e legibilidade.

2. Coesão lógica. Nesse tipo de coesão os elementos internos ao módulo apresentam uma relação lógica. É considerado melhor que o caso anterior, pois os elementos possuem uma relação. Contudo os módulos executam mais de uma função.

3. Coesão clássica. Semelhante à coesão lógica por apresentar relacionamento lógico entre os elementos internos ao módulo, mas estas relações são temporalmente dependentes, suas funções precisam ser executadas seguindo um certa lógica.

4. Coesão procedural. Assemelha-se a coesão clássica por executar as funções em uma determinada seqüência, porém, neste caso estas funções pertencem a um domínio de problema.

5. Coesão comunicacional. Possui todas as características da coesão procedural, mas neste caso há uma nova característica, os elementos internos do módulo comunicam entre si por meio de dados.

6. Coesão funcional. É considerado o mais alto grau de coesão, devido aos módulos terem uma única e bem definida função.

7. Coesão infomacional. Módulos com este tipo de coesão representam a implementação de tipos abstratos de dados. Possuem em sua estrutura dados e operações, os dados são encapsulados e escondidos, e, são acessados ou modificados unicamente por meio das operações disponíveis.

2.3. Acoplamento

Segundo Sierra e Bates (2006) o termo acoplamento pode ser entendido como o grau em que uma classe conhece outra classe. Os módulos devem ser construídos objetivando o baixo acoplamento.

Glenford Myers (1975) também classificou o acoplamento considerando do pior para o melhor caso, a classificação está a seguir.

1. Acoplamento por conteúdo. É o pior tipo de acoplamento entre módulos, este tipo de acoplamento ocorre quando um módulo faz referência direta ao conteúdo de outro módulo. O acesso ocorre sem a utilização das interfaces do módulo, prejudicando a segurança dos dados do módulo e dificultando alterações futuras no módulo.

2. Acoplamento por dado comum. Esse tipo de acoplamento ocorre quando uma mesma estrutura de dados é compartilhada por diversos módulos e cada um deles possui sua própria interpretação.

3. Acoplamento externo. Muito semelhante ao acoplamento por dado comum, com o diferencial de que vários módulos acessam um elemento comum declarado externamente.

4. Acoplamento de controle. Esse tipo de acoplamento ocorre quando um módulo é invocado e recebe um parâmetro que determina seu fluxo de execução.

5. Acoplamento por referência (Stamp). Ocorre quando uma área de dados comum não declarada globalmente é acessada por mais de um módulo via passagem de parâmetros por referência.

6. Acoplamento por informação. O acoplamento entre os módulos se dá pela chamada de rotinas com a passagem de parâmetros por valor, e, estes parâmetros não podem ser determinantes para a execução do fluxo interno do módulo.

7. Desacoplado. É considerado desacoplado se não possui nenhum tipo de comunicação entre os módulos.

2.4. Critérios, Regras e Princípios para Desenvolvimento de Software Modular

Conforme Ferreira (2006) um projeto para desenvolvimento de software deve ser dirigido à obtenção do maior grau de modularidade possível, pois proporciona melhor qualidade ao software, além de diminuir o custo da manutenibilidade e e aumentar a reusabilidade.

Segundo relatos de Santos (2006), um módulo bem projetado deve possuir uma única finalidade e uma interface apropriada com outros módulos. Assim sendo, ele é reutilizável, facilmente acoplado a outros programas, e tendo boa manutenibilidade, é capaz de sofrer revisões sem impactar no outros módulos.

Para auxiliar na obtenção da modularidade Bertrand Meyer (1997) define critérios, regras e princípios. Para uma metodologia ser considerada modular deve atender a todos os critérios definidos. Observe, entretanto que os critérios são independentes e por isso pode ocorrer o seguinte cenário, uma metodologia é adequada a um critério e não atende a outro critério. Regras determinam o que deve ser feito para atender aos critérios. Princípios são os requisitos necessários a perfeita aplicação das regras para o atendimento dos critérios.

Os cinco critérios segundo Bertrand Meyer são descritos a seguir.

- **Decomposibilidade** (*decomposability*): este critério consiste na decomposição de um problema de software em um pequeno número de subproblemas com complexidade menor, permitindo trabalhar nos subproblemas de forma independente.
- **Composilidade** (*composability*): define a construção de novos softwares a partir da combinação de elementos padronizados independentes já existentes.
- **Inteligibilidade** (*understandability*): este critério define que os módulos devem ser entendidos separadamente sem a necessidade de conhecer os demais módulos.
- **Continuidade** (*continuity*): especifica que a arquitetura de um software permite alterações em um módulo com o mínimo possível de alterações nos demais módulos.
- **Proteção** (*protection*):este critério especifica que os módulos confinem em seu interior possíveis problemas ou situações anormais de funcionamento, propagando o mínimo de erros para os demais módulos.

Segundo Bertrand Meyer, as cinco regras para alcançar os critérios descritos anteriormente são:

- **Mapeamento direto** (*direct mapping*): esta regra determina que deve ser possível determinar um modelo para o domínio do problema a ser resolvido e este deve ser compatível com a solução encontrada.

- **Poucas interfaces** (*few interfaces*): esta regra determina que todos os módulos devem se comunicar com o mínimo possível de outros módulos. Esta regra é considerada chave para se atingir os critérios de modularidade.
- **Interfaces pequenas** (*small interfaces*): esta regra determina que se dois módulos se comunicam, a quantidade de informações trocadas entre eles deve ser o mínimo possível.
- **Interface explícita** (*explicity interfaces*): esta regra determina que se dois módulos se comunicam, este relacionamento deve estar claro no texto de ambos. Por exemplo, via exportação e importação, ou via especificadores externos.
- **Ocultação de informação** (*information hiding*): esta regra determina que dados devem estar confinados dentro do módulo. Esta regra estabelece também que informações pouco relevantes para os demais módulos ou que são usadas somente internamente devem ser ocultadas no interior de seu próprio módulo, sendo visível apenas em seu interior.

Os cinco princípios, segundo Bertrand Meyer derivados das regras apresentadas, são:

- **Unidade modular lingüística** (*the linguistic modular units principle*): este princípio expressa que módulos devem corresponder as unidades sintáticas da linguagem utilizada. A linguagem referida pode ser uma linguagem de programação, de projeto ou de especificação. No caso de linguagem de programação os módulos devem ser compilados separadamente.

- **Auto-documentação** (*the self-documentation principle*): este princípio expressa que um módulo deve possuir toda a informação sobre si em seu interior. Deve ser auto-documentado. Esta auto-documentação visa legibilidade.

- **Acesso uniforme** (*the uniform access principle*): este princípio expressa que todos os serviços prestados por um módulo devem estar disponíveis por meio de uma notação uniforme conhecida externamente. Este principio está relacionado a regra de ocultação de informação, pois os serviços podem ser

utilizados sem a necessidade de conhecimento da maneira como foi implementado.

- **Aberto-fechado** (*the open-close principle*): este princípio expressa que os módulos devem ser abertos e fechados, abertos para extensão e fechados para alteração. Pode ser feita a extensão do comportamento do módulo sem realizar alterações em seu código fonte, não impactando em outros módulos que utilizam seus serviços, propiciando assim, maior manutenibilidade.

- **Escolha única** (*single choice*): este princípio expressa que um sistema de software suporta uma lista de alternativas possíveis de execução, contudo esta lista deve ser conhecida por um e apenas um módulo.

2.5. Vantagens e Desvantagens da Programação Modular

O desenvolvimento de software via programação modular traz uma série de benefícios, mas também pode trazer problemas, pois no desenvolvimento de software composto por módulos é necessário o conhecimento das interfaces dos módulos pelos desenvolvedores e estas devem ser respeitadas. As interfaces são as responsáveis pela troca de dados entre os diversos módulos que compõe o sistema, também são responsáveis pela troca de informações com o sistema operacional, usuários e outros sistemas como os de suporte.

Staa (2000) cita algumas vantagens e desvantagens da programação modular:
Como vantagens ele destaca:

- A programação modular permite vencer barreiras de complexidade, pois problemas grandes e complexos podem ser particionados em problemas menores, possibilitando sua resolução e ao término realizar sua integração.
- Permite a distribuição do trabalho dentro da equipe, pois cada módulo pode ser desenvolvido por uma pessoa diferente, tornando possível o desenvolvimento

de vários módulos ao mesmo tempo, consequentemente reduzindo o tempo necessário para a liberação do programa.

- Permite a reutilização de módulos, módulos bem projetados muitas vezes podem ser integrados a outros programas, o que reduz o esforço de implementação, pois parte da funcionalidade requerida já está disponível através destes módulos.
- Permite as organizações desenvolvedoras de software o desenvolvimento de ativos de software que podem ser utilizados com freqüência nos programas que desenvolve, esta conclusão foi baseada na percepção de que as organizações frequentemente se especializam em um domínio de aplicação especifico, e estes ativos as permitem o desenvolvimento de novas aplicações em seu domínio de forma mais rápida tornando mais competitiva.
- Melhora o gerenciamento do processo de desenvolvimento, pois permite a criação de *baselines* de módulos, disciplinando o processo de alteração de módulos existentes, reduzindo muitos problemas que surgem com a alteração indisciplinada ou acidental de partes de programas. Melhora também a visibilidade do progresso de desenvolvimento, uma vez que a medida que novos módulos vão sendo concluídos são inseridos na *baseline* tendo uma medida de quanto evolui rumo ao objetivo.
- Possibilita o desenvolvimento incremental de programas, através da criação de sucessivas versões do programa, sendo as novas versões mais completas que as anteriores e prestando um serviço significativo.
- Permite melhorar o desempenho de programas em uma época mais adequada, pois por meio da criação de sucessivas versões dos módulos, cada nova versão possuirá um aprimoramento do desempenho o que leva a programas de bom desempenho. O bom projeto de módulos permite a substituição destes por módulos que prestem o mesmo serviço, contudo com melhor desempenho, não afetando o funcionamento do programa.
- Finalmente reduz o tempo de compilação, pois não será necessário recompilar todos os módulos, sendo necessário à compilação de novos módulos ou módulos que sofreram alterações.

Como desvantagens, Staa disponibiliza algumas perguntas, a maioria de difíceis respostas.

- Quais as diversas formas como são apresentados os módulos?
- Como realizar a partição de programas em módulos?
- De que maneira realizar a partição de módulos de maneira que uma parte significativa destes módulos possa ser reutilizável de forma efetiva?
- Como deve ser feita a especificação de módulos?
- Quais as formas de garantir a qualidade de um módulo?
- Como garantir a qualidade dos componentes, composição de vários módulos?
- Como viabilizar e gerenciar o desenvolvimento de vários módulos de forma independente, por diferentes pessoas?
- Como criar um módulo de maneira que seja facilmente alterado sem impactar em outros módulos, mesmo por pessoas que não participaram de seu desenvolvimento?
- Como manter a coerência entre os módulos de um determinado componente após alterações que venham a ocorrer?
- Como manter a coerência dos componentes levando em conta a evolução dos módulos que o compõe?

2.6. Conclusão

Esta seção apresentou o resultado de um estudo sobre programação modular, destacando as definições de programação modular, componentes, módulos. Também foram vistos conceitos de coesão e acoplamento, e como a coesão e acoplamento são importantes para construção de software de boa qualidade. Foi destacada a classificação de Myers (1975) para os tipos de coesão e acoplamento, avaliando do pior para o melhor caso.

Também foi visto nesta seção os critérios, regras e princípios definidos por Bertrand Meyer (1997), estes podem ser utilizados com o intuito de gerar um produto verdadeiramente modular, ou ainda avaliar se um produto existente pode ser considerado modular. Apresentamos a relação existente entre os conceitos de critérios, regras e princípios, bem como a utilização dos princípios para se atender as

regras, o atendimento das regras para se obter os critérios, a satisfação dos critérios para obtenção de um produto verdadeiramente modular.

Por último foi visto as principais vantagens e desvantagens ou questionamentos de se construir software utilizando a programação modular, Staa (2000) cita algumas vantagens como redução da complexidade pela divisão do problema em problemas menores, melhora na divisão do trabalho dentro de equipes, aumento da reusabilidade e manutenibilidade gerando economia de recursos e tempo dentre outros. Porém explicita também alguns questionamentos relacionados a programação modular como por exemplo formas de apresentação de módulos, particionamento de programas em módulos, dentre outros.

Esta seção permite ao leitor entender melhor a programação modular e suas principais características além de maneiras de se obter verdadeiramente softwares considerados modulares e com melhor qualidade. Com esses conceitos em mente, a Seção 3 apresenta o resultado de um estudo sobre qualidade de software, seus conceitos, e formas de se obter a qualidade.

3. QUALIDADE DE SOFTWARE

Esta seção tem como objetivo apresentar o resultado de um estudo sobre qualidade de software, os conceitos relacionados, a importância da qualidade e como associar o controle da qualidade sobre o processo de desenvolvimento de software, conceitos relacionados a controle de qualidade e garantia da qualidade.

A avaliação da qualidade de produtos de software deve ser considerada um fator de extrema importância, isso devido a crescente utilização de computadores nas diversas áreas de aplicação, sendo considerada um fator de destaque para o sucesso de negócios e muitas vezes crítico para a segurança humana. Com base nestas considerações é de suma importância o desenvolvimento de software de alta qualidade, mas para obtenção de produtos de qualidade adequada devem ser considerados aspectos cruciais como a especificação e avaliação da qualidade do produto. NBR 9126 (2003).

Staa (2000) afirma que a qualidade de software não deve ser tratada como assunto secundário devido à dependência existente, e cada vez é maior a presença de serviços oferecidos por software e que afetam a vida de todos, direta ou indiretamente.

> Qualidade é palavra de ordem em quase todos os segmentos da produção humana nos tempos atuais, talvez pelo caráter competitivo que a maior parte das relações possui.
> A qualidade do trabalho pode significar o sucesso de um profissional, assim como a qualidade dos produtos e serviços pode significar o sucesso de uma organização. Não é diferente no caso da produção de software. Em um contexto de mercado competitivo e forte demanda por sistemas cada vez mais complexos, a qualidade de um software é o fator determinante para o seu sucesso. Assim, é importante a existência de formas de se avaliar a qualidade de software. (FERREIRA, 2006, p. 23)

3.1. Definições de Qualidade de Software

Staa (2000) define a qualidade de um artefato como uma totalidade de atributos a serem satisfeitos em um determinado nível de forma que as necessidades de seus usuários sejam atendidas. Ele tipifica estes usuários da seguinte forma:

- usuário da aplicação: ou usuário comum, quem efetivamente faz uso da aplicação;
- desenvolvedor: profissional de informática que utiliza as especificações de um determinado artefato para criar outros artefatos que utilizem seus serviços;
- outros artefatos: artefatos que possuam interface com o determinado artefato e utilizam seu serviço;
- agressores: pessoas ou outros artefatos que o utilizem o artefato de maneira inapropriada, podendo causar falhas.

Para Myers (1975) a qualidade é descrita como uma combinação de vários fatores, tais como rapidez, facilidade de uso, confiabilidade, modularização, boa estruturação, facilidade de leitura e outros. Myers diz que estes fatores descrevem dois tipos diferentes de qualidade. Fatores como facilidade de uso ou rapidez, podem ser percebidos pelos usuários em um produto de software, ou seja, são características percebidas pelos usuários do sistema e devido a este fato são denominados como fatores externos de qualidade.

Outros fatores como modularidade, legibilidade e manutenibilidade são percebidos pelos profissionais que desenvolvem o software e devido a isto são denominados fatores internos de qualidade de software.

3.2. Fatores Internos e Externos de Qualidade de Software

Do ponto de vista de qualidade de software, Myers (1975) considera dois aspectos ou características, denominados respectivamente de fatores internos e fatores externos, descritos a seguir.

Os fatores internos de qualidade de um software podem ser entendidos como a qualidade sob o ponto de vista de seus produtores, ou seja, profissionais de informática como, analistas de sistemas, programadores, administradores de banco de dados, arquitetos, gerentes e outro. São eles: modularidade, legibilidade, e manutenibilidade.

1. Modularidade: é a construção de software via módulos independentes prestando serviços distintos.

2. Legibilidade: é a facilidade de entendimento.

3. Manutenibilidade: é a medida de facilidade de realizar alterações ou inserção de novas funcionalidades no produto, é um fator que impacta no custo de uma aplicação.

Fatores externos podem ser entendidos como a qualidade sob o ponto de vista de quem usa o software, ou seja, usuários em geral:

1. Correção: é a propriedade de o software ser correto, realizar sua função conforme definida pelos requisitos e especificação.

2. Robustez: é a capacidade de um software de reagir de forma adequada mesmo em situações anormais.

3. Extensibilidade: é a facilidade de adaptação do software a mudanças, seja por modificações na especificação de seus requisitos ou inserção de novas funcionalidades.

4. Reusabilidade: é a facilidade de reutilização do software, seja parcialmente ou em sua totalidade, por outros produtos de software.

5. Compatibilidade: é a capacidade de interação com outros produtos de software.

6. Eficiência: é a propriedade de o software fazer bom uso de recursos de hardware.

7. Portabilidade: é a facilidade que o software tem de funcionar bem em diferentes plataformas de hardware ou software.

8. Verificabilidade: é a facilidade de se preparar procedimentos de aceitação, que verificam a conformidade do software com seus requisitos.

9. Integridade: é a capacidade de proteção dos dados contra acesso não autorizado.

10. Facilidade de Uso: está associado a usabilidade, é a facilidade de aprendizado e utilização do software.

3.3. Modelo de Avaliação de Qualidade de Software

A norma NBR 9126 (2003) descreve um modelo de qualidade de produto de software para definição de metas de qualidade de produto de software final ou intermediário. Esse modelo é composto por duas partes, a primeira parte é sobre qualidade interna e qualidade externa, e a segunda parte é sobre qualidade em uso. Este modelo descreve características e subcaracterísticas que podem ser utilizadas como uma lista de verificação de tópicos relacionados à qualidade. Contudo, na prática não é possível medir todas as subcaracterísticas da qualidade interna e externa para software de grande porte, e também não é prático medir a qualidade em uso para todos os cenários de qualidade em uso.

3.3.1. Qualidade Interna e Qualidade Externa

Segundo a NBR 9126 (2003) a primeira parte do modelo de qualidade, sobre qualidade interna e externa, especifica seis características, como pode ser visto na Figura 1. Estas características são dividas em subcaracterísticas que são percebidas externamente quando o software é utilizado como parte de um sistema computacional. As subcaracterísticas são resultantes de atributos internos do software. A NBR 9126 (2003) limita-se a apresentar o modelo de qualidade interna e externa de software em nível de subcaracterísticas.

A norma NBR 9126 (2003) define a qualidade externa como o conjunto de características do produto de software do ponto de vista externo, é entendida como a qualidade do produto no momento que ele é executado, a medição e avaliação do produto de software é feito tipicamente em um ambiente simulado. É conveniente que todos os defeitos sejam localizados e corrigidos durante os testes. Defeitos na arquitetura do software ou aspectos básicos do software são difíceis de ser corrigidos considerando que a base do projeto permanece inalterada após os testes.

A qualidade interna é definida como o conjunto de características do produto de software do ponto de vista interno, estas características são medidas e avaliadas do ponto de vista interno. Os detalhes da qualidade do produto podem ser melhorados durante a implementação do código a partir de revisão e testes, contudo, a natureza fundamental da qualidade do software permanece inalterada a não ser que seja reprojetada.

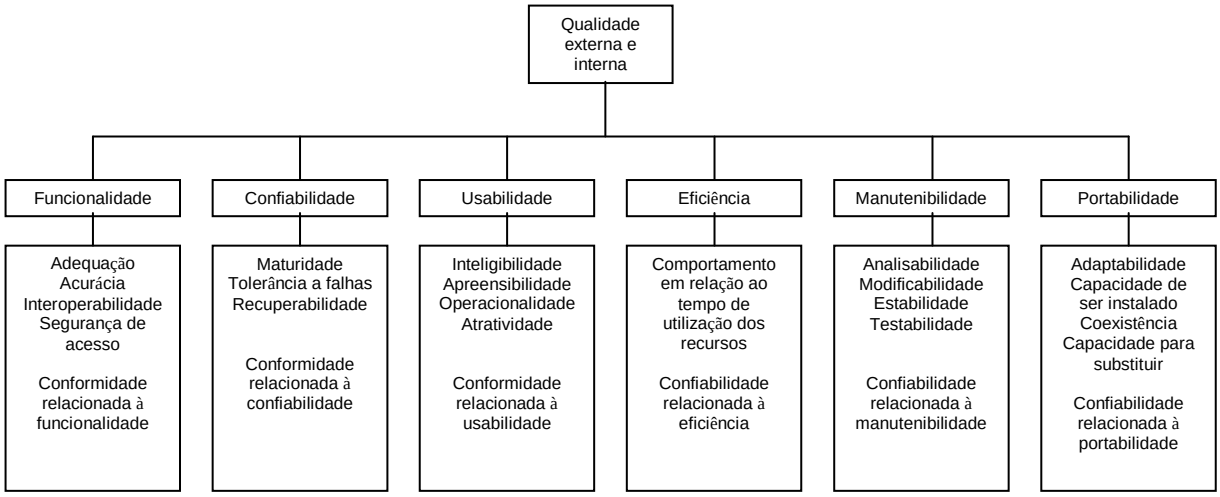

Figura 1 – Modelo de qualidade para qualidade interna e externa
Fonte: NBR 9126 (2003) p.7.

As definições dadas pela NBR 9126 (2003) a cada característica e cada subcaracterística da qualidade interna e qualidade externa de um produto de software são listadas a seguir.

- Funcionalidade: capacidade de fornecer funções que atendam as necessidades explícitas e implícitas quando estiver em uso sob determinadas condições.
    - Adequação: capacidade de fornecer funções adequadas para as tarefas e objetivos dos usuários especificados.
    - Acurácia: capacidade de prover resultados ou efeitos corretos com a precisão necessária, conforme acordado.
    - Interoperabilidade: capacidade do produto de software de interagir com outros sistemas.
    - Segurança de acesso: capacidade de proteger informações e dados de acesso de pessoas ou sistemas não autorizados, e oferecer o serviço a pessoas ou sistemas autorizados.
    - Conformidade relacionada à funcionalidade: capacidade de estar de acordo com normas, convenções ou regulamentações legais e prescrições análogas relacionadas à funcionalidade.

- Confiabilidade: capacidade de manter um nível de desempenho determinado, quando estiver em uso sob condições determinadas.

- Maturidade: capacidade de se evitar falhas decorrentes de defeitos no software.
- Tolerância a falhas: capacidade de manter um nível de desempenho determinado mesmo com a ocorrência de defeitos no software ou violação de sua interface especificada.
- Recuperabilidade: capacidade de recuperação do nível de desempenho determinado e recuperação de informações ou dados afetados pela falha.
- Conformidade relacionada à confiabilidade: capacidade de estar de acordo com normas, convenções ou regulamentações legais e prescrições análogas relacionadas à confiabilidade.

- Usabilidade: capacidade de ser compreendido, aprendido, operado e ser esteticamente bem definido para o usuário, quando estiver em uso sob condições determinadas.
  - Inteligibilidade: capacidade de permitir ao usuário a compreender se o software é adequado e permitir ao usuário o entendimento de como o software pode ser utilizado para determinadas tarefas.
  - Apreensibilidade: capacidade de permitir ao usuário aprender sobre a sua aplicação.
  - Operacionalidade: capacidade de permitir ao usuário operá-lo e controlá-lo.
  - Atratividade: capacidade de ser atraente aos usuários, bem definido esteticamente.
  - Conformidade relacionada à usabilidade: capacidade de estar de acordo com normas, convenções ou regulamentações legais e prescrições análogas relacionadas à usabilidade.

- Eficiência: capacidade de apresentar desempenho adequado, relativo a carga de recurso usados e quando estiver em uso sob condições determinadas.
  - Comportamento em relação ao tempo: capacidade de fornecer tempos de reposta, processamento e taxas adequadas quando estiver em uso sob condições determinadas.

- Utilização de recursos: capacidade de uso de tipos e quantidades adequados de recursos, quando estiver sendo executado sob condições determinadas.
- Conformidade relacionada à eficiência: capacidade de estar de acordo com normas, convenções ou regulamentações legais e prescrições análogas relacionadas à eficiência.

- Manutenibilidade: capacidade de ser modificado, as modificações incluem correções, melhorias ou adaptações oriundas de alterações no ambiente, requisitos ou especificações funcionais.
  - Analisabilidade: capacidade de permitir o diagnóstico de deficiências ou motivos de falhas, e também, a localização de partes a serem modificadas.
  - Modificabilidade: capacidade de permitir que uma alteração deterinada seja realizada.
  - Estabilidade: capacidade de evitar efeitos indesejáveis provenientes de alterações realizadas.
  - Testabilidade: capacidade de validação do software quando ocorrerem alterações em sua estrutura.
  - Conformidade relacionada à manutenibilidade: capacidade de estar de acordo com normas, convenções ou regulamentações legais e prescrições análogas relacionadas à manutenibilidade.

- Portabilidade: capacidade do produto de software de ser transferido de um ambiente para outro.
  - Adaptabilidade: capacidade de adequação a diferentes ambientes determinados sem a necessidade de consumo de recursos além dos especificados.
  - Capacidade para ser instalado: capacidade de ser instalado em um ambiente determinado.
  - Coexistência: capacidade de coexistir com outros produtos de software independentes, no mesmo ambiente e compartilhando recursos comuns.

- Capacidade para substituir: capacidade de substituir outro produto de software que possua o mesmo propósito e esteja instalado no mesmo ambiente.
- Conformidade relacionada à portabilidade: capacidade de estar de acordo com normas, convenções ou regulamentações legais e prescrições análogas relacionadas à portabilidade.

3.3.2. Qualidade em Uso

De acordo com a NBR 9126 (2003) a segunda parte do modelo de qualidade, sobre qualidade em uso, especifica quatro características, como visto na Figura 2. A NBR 9126 (2003) limita-se a apresentar o modelo de qualidade em uso em nível de características.

A qualidade em uso é definida pela norma NBR 9126 (2003) como sendo a qualidade do ponto de vista do usuário, fase em que o produto é utilizado em um ambiente e em um contexto de uso especificados. A qualidade em uso mede o quanto os usuários podem atingir seus objetivos e não a propriedades do software em si.

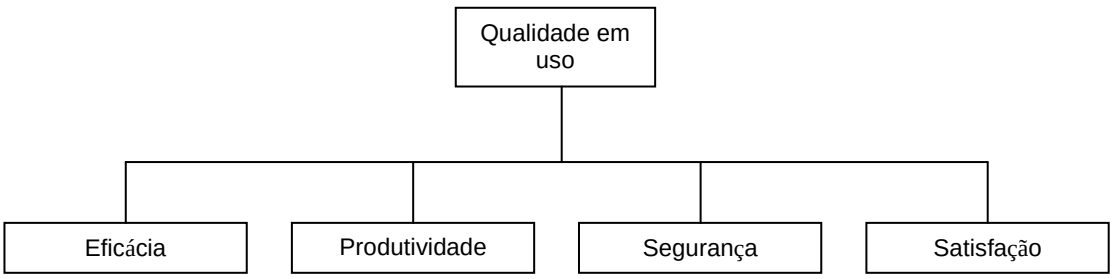

Figura 2 – Modelo de qualidade em uso
Fonte: NBR 9126 (2003) p.11.

As definições dadas pela NBR 9126 (2003) a cada característica da qualidade em uso de um produto de software são listadas a seguir.

- Qualidade em uso: capacidade de permitir a usuários determinados, atingir metas como eficácia, produtividade, segurança e satisfação, sob condições de uso determinadas.

- Eficácia: capacidade de permitir aos usuários atingir metas de acurácia e completude, sob um contexto de uso determinado.
- Produtividade: capacidade de permitir aos usuários o emprego adequado de recursos em relação a eficácia obtida, sob um contexto de uso determinado.
- Segurança: capacidade de apresentar níveis aceitáveis de riscos de danos a pessoas, negócios, software, propriedades ou ambiente, sob um contexto de uso especificado.
- Satisfação: capacidade de satisfazer aos usuários, sob um contexto de uso espcificado.

3.3.3. Requisitos de Qualidade e Ciclo de Vida de Software

Segundo a NBR 9126 (2003) as visões de qualidade interna, qualidade externa e qualidade em uso se alteram durante o ciclo de vida do software. Como exemplo, a qualidade especificada como requisito no início do ciclo de vida de um software é uma visão predominantemente do ponto de vista externo ou do usuário, diferindo da qualidade do produto intermediário, como também da qualidade na fase de projeto, que é uma visão predominantemente do ponto de vista da qualidade interna ou dos desenvolvedores. As tecnologias empregadas para realizar a especificação e validação de qualidade devem levar em conta as perspectivas de qualidade, com o objetivo de gerenciar a qualidade de maneira adequada em cada fase do ciclo de vida.

A Figura 3 mostra a existência de diferentes visões da qualidade do produto e suas métricas em diferentes estágios do ciclo de vida do software.

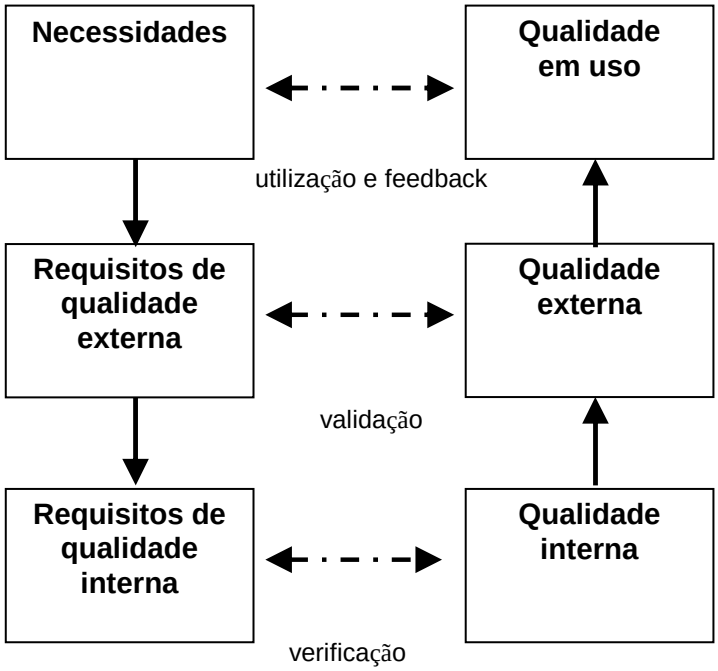

Figura 3 – Qualidade no ciclo de vida do software
Fonte: NBR 9126 (2003) p.5.

Com base nessa figura, a NBR 9126 (2003) conclui que os requisitos de qualidade não podem ser completamente definidos antes de iniciar o projeto e também entender as necessidades reais dos usuários com o máximo nível de detalhe possível, pois serão representadas nos requisitos.

3.4. Garantia da Qualidade

Para Staa (2000) a qualidade de um artefato poderá ser garantida somente se for previamente definida anteriormente a sua produção. Pois sem o conhecimento prévio do grau de qualidade a ser atingido, qualquer resultado será considerado bom ou mesmo ruim devido à inexistência de parâmetros de comparação.

Staa (2000) afirma ainda que a garantia da qualidade de um artefato não poderá ser obtida por meio do controle da qualidade, destacando testes, pois o controle da qualidade serve apenas para apontar o nível de qualidade que um determinado artefato possui. Para ele a única forma de se obter a qualidade é guiar o desenvolvimento ou a manutenção de forma a atingir, por construção, as metas de

qualidade previamente estabelecidas, removendo também os problemas identificados por meio do controle da qualidade.

Staa (2000) apud Humphrey (1989), Paulk et al (1995), Fiorini et al (1998) especifica que o propósito da garantia da qualidade é assegurar que a probabilidade de um artefato produzido tenha a qualidade efetiva melhor que a qualidade requerida.

Staa (2000) resume como condições necessárias para atingir um nível de qualidade satisfatória os seguintes itens:

- Desenvolvedores e mantenedores devem ter uma atitude coerente ao desenvolvimento de artefatos de boa qualidade.
- Os profissionais envolvidos devem ter boa formação, treinamento e proficiência adequada para realização das atividades e papéis que irão exercer.
- Os artefatos devem possuir especificação adequada e confiável antes de serem desenvolvidos.
- Deve ser utilizado um processo de desenvolvimento que seja capaz de produzir, por construção, os artefatos com o nível de qualidade especificado.
- A qualidade dos artefatos e componentes deve ser controlada de forma sistemática, considerando um conjunto definido e documentado de padrões, critérios para avaliação e requisitos contidos na especificação dos artefatos.
- Deve ser utilizado um processo para controle de alterações capaz de conduzir a soluções completas e corretas, seja de problemas identificados pelo controle da qualidade ou requisições de alterações recebidas e registradas.

E cita também exemplos de itens a serem verificados de forma a garantir a qualidade do produto, como:

- Se as normas, padrões, práticas e convenções são obedecidas, tanto com relação à qualidade dos artefatos, como também em relação ao processo.
- Se a documentação é gerada, atualizada e está disponível para consulta e é coerente, ou seja, seu conteúdo é pertinente.
- Se são realizadas revisões e são adequadamente conduzidas.
- Se os testes são especificados, realizados e conduzidos adequadamente.

- Se os problemas identificados durante o processo de controle da qualidade são registrados e acompanhados até a completa resolução do problema. Não se deve corrigir sem avaliação, pois a solução pode ser incompleta ou feita repetidas vezes.
- Se ocorre a coleta dados com objetivo de utilizar para melhorar a qualidade ou processo de software.
- Se são utilizados padrões, métodos, técnicas e ferramentas adequadas ao artefato a ser desenvolvido.
- Se, após aprovação os artefatos são armazenados em bibliotecas controladas (*baseline*).
- Se existe um sistema de backup.
- Se o pessoal envolvido no projeto é habilitado a utilizar as ferramentas, padrões e processos estabelecidos.

Para Paula Filho (2009), as ações de garantia da qualidade na produção de software devem ser utilizadas de maneira preventiva em relação a problemas da qualidade do produto que poderiam surgir. Deve-se buscar a inserção de mecanismos preventivos nos processos a fim de reduzir o número de defeitos que podem ser inseridos no decorrer do projeto.

Paula Filho (2009) apud CMMI define os seguintes critérios para a garantia da qualidade de produtos ou processos, vide Quadro 1.

Quadro 1 – Objetivos para a garantia da qualidade de produtos ou processos

Objetivos para garantia da qualidade de produtos ou processos
Avaliar de forma objetiva os processos e produtos de trabalho, contra as respectivas descrições de processo, padrões e procedimentos.
Identificar e documentar problemas de não conformidade.
Garantir que estes problemas sejam tratados até a resolução completa.
Fornecer informação ás equipes técnicas e aos gerentes sobre questões de qualidade

Fonte: Adaptado de Paula Filho, p.457.

3.5. Conclusão

Nesta seção foram abordados os principais conceitos sobre qualidade de produtos de software, dentro da visão de autores como Ferreira (2006), Meyer (1997), Staa (2000), Paula Filho (2009). Foi apresentada a contextualização de qualidade definida por Meyer dividindo os aspectos de qualidade em dois grupos distintos, fatores de qualidade interna e externa, como pontos de vista de percepção da qualidade para usuários e profissionais envolvidos no desenvolvimento.

Foi apresentado o modelo de qualidade proposto pela NBR 9126 (2003). Este modelo sugere a divisão de aspectos de qualidade a serem avaliados como grupos de características e subcaracterísticas de qualidade interna e qualidade externa e ainda qualidade em uso. Este modelo pode ser utilizado para medir os níveis de qualidade de um produto de software como uma lista de itens a serem verificados.

Por último foi visto aspectos sobre garantia da qualidade que descreve as formas de se alcançar a qualidade almejada de um determinado produto, e, foram descritas as condições necessárias para obtenção da qualidade e itens a serem verificados como forma de garantia.

4. CONCLUSÃO

Esta seção tem como objetivo apresentar uma conclusão sobre os itens apresentados anteriormente sobre a programação modular e os fatores que impactam na qualidade do produto de software.

O estudo realizado nesta monografia possibilitou observar a importância do desenvolvimento de softwares de boa qualidade, em função da dependência cada vez maior desses produtos em diversas áreas de aplicação e as conseqüências que a baixa qualidade destes produtos podem gerar. Estes produtos podem ser considerados críticos para o sucesso de negócios e organizações e podem também ser críticos para a segurança humana.

Os itens observados sobre a programação modular demonstram que esta técnica pode ser muito eficaz na obtenção de software de qualidade, devido a fatores como manutenibilidade e reusabilidade. A programação modular é considerada conforme Staa (2000), uma base para o desenvolvimento de programas de forma organizada, gerando programas estáveis e flexíveis, aumentando a confiabilidade, manutenibilidade e reusabilidade.

Com o tópico sobre vantagens da utilização da programação modular da Seção 2 foram percebidos uma série de vantagens ao se empregar esta técnica, tais como, a redução da complexidade dos problemas por meio da divisão em problemas menores, melhor distribuição do trabalho, reutilização, aumento da manutenibilidade, melhor gerenciamento do processo de software, aspectos que impactam na qualidade dos produtos e nos custos.

A seguir uma lista resumindo os principais benefícios da utilização da programação modular contidos no trabalho:

- Software estável e flexível: devido à construção por unidades independentes.
- Maior confiabilidade: a capacidade de manter o desempenho mesmo em condições de falhas devido a maturidade dos módulos que compõe o produto.

- Manutenibilidade: devido ao bom relacionamento entre os módulos e relacionamento interno dos módulos.
- Reusabilidade: devido a existência de módulos independentes prestando serviços via uma interface bem definida, este módulos poderão ser utilizados em outra aplicações que necessitem de seu serviço.
- Melhor gerenciamento do processo: facilita o gerenciamento do processo com a criação de *baselines*, nas quais são inseridos módulos homologados garantindo assim a qualidade pela maturidade destes módulos.
- Melhor distribuição do trabalho: facilita a distribuição do trabalho devido a possibilidade de várias pessoas trabalhar sobre os vários componentes de um software paralelamente, tendo com o foco unicamente a funcionalidade que devem possuir.
- Obtenção de produtos de software com qualidade obtida por construção: via a construção de módulos independentes devido a redução da complexidade do problema em problemas menores é possível desenvolver software de boa qualidade por construção.
- Redução de custos: facilita a manutenibilidade que pode ser considerada um dos principais aspectos de custo de um produtos de software.
- Atendimento das métricas de avaliação da qualidade de software: atende as métricas de funcionabilidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

REFERÊNCIAS

FERREIRA, Kécia A. M. **Avaliação de Conectividade em Sistemas Orientados por Objetos.** Dissertação de Mestrado, 2006, DCC – UFMG.

GOMES, Nelma da Silva. **Qualidade de Software – Uma Necessidade.** Disponível em: "http://www.fazenda.gov.br/ucp/pnafe/cst/arquivos/Qualidade_de_Soft.pdf". Acesso em: 24 jul 2010.

MARTINS, Vanessa Nogueira. **Padrões de Projeto e Princípios de Projeto.** Monografia de Pós-graduação, 2006, DCC – UFMG.

MEYER, Bertrand. **Object-oriented Software Construction.** 2.ed. – Estados Unidos. Prentice Hall, 1997. ISBN 01-362-9155-4.

MYERS, Glenford J. **Reliable software through composite design.** Nova York: Petrocelli/Charter, 1975. ISBN 0-88405-284-2.

NBR ISO/IEC 9126-1. **Engenharia de Software – Qualidade de Produto**. 2003.

OLIVEIRA, Karina da Silva et al. **Abordagens de Reúso de Software no Desenvolvimento de Aplicações Orientadas a Objetos.** In: Escola Regional de Informática, 2006, Bandeirantes. Anais do ERI-PR. 2006. Disponível em : "http://inf.cp.utfpr.edu.br/ligia/papers/reuso.pdf". Acesso em: 25 jul 2010.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software: fundamentos, métodos e padrões.** 3.ed. – Rio de Janeiro. LTC, 2009. ISBN 978-85-216-1650-4.

SANTOS, Karla Costa. **Princípios de Programação Modular.** Monografia de Pós-graduação, 2006, DCC – UFMG.

SIERRA, Kathy; BATES, Bert. **SCJP: Certificação Sun para Programador Java 5 - Guia de Estudo.** Editora Alta Books, 2006. ISBN: 85-760-8133-4.

STAA, Arndt von. **Programação modular – Desenvolvendo problemas complexos de forma organizada e segura.** – Riod de Janeiro. Editora Campus, 2000. ISBN 85-352-0608-6.

YOURDON, Edward. (1999). **Análise e projeto orientados a objetos.** Makron Books, São Paulo.