INFORMÁTICA 82

XV CONGRESSO NACIONAL DE INFORMÁTICA
II FEIRA INTERNACIONAL DE INFORMÁTICA

ANAIS

INFORMÁTICA 82

XV CONGRESSO NACIONAL DE INFORMÁTICA
II FEIRA INTERNACIONAL DE INFORMÁTICA

ANAIS

QUALIDADE DE SOFTWARE

Arndt von Staa
Departamento de Informática da Pontifícia Universidade Católica
Rio de Janeiro, RJ

Software de qualidade é útil, utilizável, monitorável, evolutível e rentável. Para que o software venha a apresentar níveis de qualidade satisfatórios, não basta boa vontade por parte da equipe de desenvolvimento. É necessário um esforço consciente, continuado e bem fundamentado em determinar i) qual o nível de qualidade a ser alcançado antes mesmo de iniciar o desenvolvimento, ii) controlar o processo de definição e construção, bem como iii) controlar a qualidade dos produtos resultantes destas atividades. Neste artigo justificamos a necessidade de uma maior preocupação com a qualidade de software, como proceder para alcançá-la, como especificá-la e, finalmente, como controlar a qualidade de software.

Palavras-chave: Avaliuação da qualidade; Engenharia de Software; Fatores de qualidade; Funções de normalização; Métricas de qualidade de software; Objetivos de qualidade; Predição de qualidade; Qualidade de software; software.

1. INTRODUÇÃO

É lugar comum reclamar-se da baixa qualidade do software e do elevado custo de seu desenvolvimento. No entanto, pouco é dito com relação a o que vem a ser qualidade de software, como especificá-la, como projetar e construir sistemas para que tenham níveis satisfatórios de qualidade, e como avaliar os resultados da construção e uso quanto à sua qualidade realmente percebida.

Com o crescimento da abrangência de atuação do software, e com o consequente crescimento da complexidade deste software, mais difícil tem-se tornado assegurar níveis mínimos de qualidade. À medida que nos tornamos mais dependentes dos sistemas de software, mais nefastas se tornam as conseqüências de baixa qualidade, portanto, mais qualidade exigimos do software.

O objetivo deste artigo é definir o que se entende por qualidade de software, introduzir os conceitos correspondentes, e delinear um método de especificação e controle de qualidade de software.

A literatura corrente descreve diversos modelos para especificação e controle da qualidade de sistemas de software. Estes vão dos mais limitados em escopo, a modelos hierárquicos complexos. Os métodos utilizados para controlar a qualidade também variam muito. Vemos assim técnicas de avaliação baseadas em prova de correção, em inspeções ("walk throughs"), em testes sistemáticos e abrangentes, em controle estatístico da qualidade, etc. Este material todo daria para encher vários livros. Optamos aqui por somente delinear um modelo de definição e controle de qualidade simples e abrangente. Ao leitor interessado em aprofundar-se neste assunto, recomendamos consultar a bibliografia indicada no final deste artigo.

Como em qualquer ramo industrial, a qualidade do produto é um objetivo de projeto, sendo raras as ocasiões em que qualidade pode ser incorporada ao produto de forma não consciente ou, então, após o produto ter sido construído. Assim, ao desenvolvermos software, devemos sempre ter em mira o objetivo de qualidade previamente estabelecido. Sem conhecer este objetivo e sem trabalhar conscientemente para alcançá-lo, é virtualmente impossível conseguirmos ter certeza quanto à qualidade do software sendo construído.

2. NECESSIDADE DA QUALIDADE DE SOFTWARE

A necessidade da qualidade de programas há muito já vem sendo notada (Boehm 75). Qualquer texto corrente que trate de engenharia de software, de técnicas e/ou de métodos de programação, cita a necessidade de melhorar a qualidade dos programas (sistemas)

desenvolvidos. Infelizmente muitos tendem a ver o problema de qualidade exclusivamente como um problema da correção dos algoritmos empregados. Apesar de, sem dúvida, este ser um problema crucial, não bastam os algoritmos utilizados estarem formalmente corretos para que tenhamos um software de boa qualidade.

Além de estarem corretos, os programas devem produzir resultados suficientemente precisos dentro de limitações de espaço de memória, tempo de resposta, custo de criação e de operação. Para alcançarmos boa qualidade devemos, pois, procurar soluções simples e adequadas ao problema em apreço. Note que o enfoque é o de selecionar um algoritmo correto e suficientemente eficiente e preciso para o problema a ser resolvido. Dito de outra forma, desempenho não é problema da linguagem de programação utilizada, e sim um problema do projeto, dos algoritmos e das estruturas de dados utilizadas.

Temos agora um sistema correto, preciso e eficiente. Porém este sistema será pouco conveniente, se não for também suficientemente flexível de modo a poder adaptar-se a um conjunto grande de condições ambientais e/ou de uso diferentes. Dito de outra forma, assegurar a correção e eficiência pode levar a software cujo custo de manutenção e operação seja demasiadamente alto. O resultado disto é um prejuízo considerável, quando o que queríamos era justamente economia...

Assumamos agora que o nosso software foi construído utilizando algoritmos corretos, suficientemente precisos e obedecendo a regras de construção que o tornem muito flexível. Porém para utilizar o software temos que fornecer um conjunto grande de parâmetros capazes de descrever as condições reais de cada uso específico. É evidente que este software satisfaz aos requisitos acima estabelecidos. Porém, para que possamos utilizá-lo, precisamos sempre recorrer ao seu manual do usuário. Dito de outra forma, a tentativa de assegurar flexibilidade, pode levar a software cujo custo de construção, manutenção e uso seja demasiadamente alto. Observe que atingimos um impasse: se não for flexível é claro, se for demais também o será. Isto fortalece a necessidade de *projetar-se* o nível de qualidade a alcançar antes de partir para a construção.

Podemos concluir, então que qualidade de software é um conjunto de propriedades do software a serem satisfeitas em determinado grau, de modo a satisfazer as necessidades do usuário deste software. Cabe salientar os seguintes pontos:

i) qualidade depende de um conjunto de propriedades. Algumas destas são frequentemente mecionadas, tais como correção, confiabilidade, flexibilidade, etc. Já outras são pouco mencionadas, tais como operabilidade, facilidade de preparação dos dados, facilidade de interpretação dos resultados, rentabilidade, auditabilidade, etc.

ii) as propriedades são avaliáveis, existindo um grau mínimo a ser atingido para que o software em apreço seja de boa qualidade. Deve existir, portanto, um sistema de métricas que permita medir ou avaliar as diversas propriedades.

iii) a qualidade do software depende da satisfação do usuário com o serviço (resultados) do software e com a facilidade de seu uso. Aqui cabe salientar a existência de diversos tipos de usuários (operador, manutenedor, direto, indireto, auditor, outro programa ou sistema, etc.) sendo que a qualidade depende da satisfação das necessidades e expectativas de todos estes usuários. Diferentes usuários, ainda que pertencentes a uma mesma categoria, podem observar graus de qualidade diferentes.

iv) as propriedades podem ser conflitantes entre si. É comum, por exemplo, flexibilidade ser conseguida às custas de um menor desempenho. Torna-se necessário então definir os graus mínimos de aceitabilidade.

De uma forma genérica podemos dizer que um software de boa qualidade produz resultados úteis e confiáveis na oportunidade certa; é ameno ao uso; é mensurável e auditável; é corrigível, modificável, e evolutível; opera em máquinas e ambientes reais: foi desenvolvido de forma econômica e no prazo estipulado; e opera com economia de recursos.

3. GARANTIA DE QUALIDADE

Nesta seção examinaremos como proceder para que tenhamos certeza de estarmos desenvolvendo software de qualidade satisfatória consumindo o mínimo de recursos para desenvolver e operar este software.

Como já havíamos mencionado, a qualidade do software depende, em última análise, do serviço (resultados) por ele prestado e da facilidade com que conseguiremos obter este serviço. Ou seja, do ponto de vista do usuário, não importa como foi construído o software desde que faça o esperado sem criar contratempos desnecessários. Cabe observar que os objetivos de serviço do software variam no decorrer do tempo. Cabe à manutenção de software a tarefa de assegurar a contínua concordância entre estes objetivos e o sistema em si. É claro que a manutenção do software deverá assegurar também a manutenção, ou melhoria, dos níveis de qualidade deste software. Cabe enfatizar que entendemos como software não somente o código, e sim todo o conjunto formado por código, dados, documentos, etc., necessários para utilizar, manter, avaliar e entender o sistema e os resultados por ele produzidos. Cabe à manutenção assegurar a consistência entre estas diversas formas de representação do software.

A qualidade do serviço de um software é, portanto, um reflexo da qualidade do software e do cuidado com que é utilizado e mantido. Quanto menor o cuidado de uso necessário para obter resultados úteis, mais robusto e utilizável é o software.

Por sua vez, para que o software possua um nível de qualidade satisfatório, é necessário que tenha sido desenvolvido com o intuito de alcançar este nível. Devem, pois, existir especificações que determinem o que é considerado como sendo qualidade satisfatória, devendo, ainda, ser empregado um processo de construção que assegure atingir-se este nível de qualidade estipulado.

A qualidade de software está baseada nas seguintes noções:

i) expectativa de qualidade – estabelece os requisitos de qualidade do serviço a ser prestado pelo software, do ponto de vista de seus diversos usuários;

ii) definição da qualidade do produto – determina a qualidade mínima dos diversos sub-produtos do desenvolvimento (projetos, programas, documentos, etc.),

de modo que, satisfazendo o produto esta qualidade definida, é viabilizada a satisfação da expectativa de qualidade do serviço prestado ;

iii) controle de qualidade – mede ou avalia o produto pronto com o intuito de verificar se está satisfazendo à sua definição de qualidade;

iv) predição da qualidade do software – estima a qualidade alcançável pelo sistema, caso venha a ser construído em acordo com as especificações e seguindo o processo de desenvolvimento determinado. A predição da qualidade do software procura determinar se a expectativa de qualidade do serviço pode ser alcançada a partir dos diversos sub-produtos do desenvolvimento do software (especificações, projetos, programas, módulos, etc.).

v) garantia de qualidade – controla o processo de construção de modo que resulte em um software satisfazendo a sua expectativa de qualidade de serviço.

A definição da expectativa de qualidade do serviço é fruto do processo de definição(especificação) do software. Este processo de definição parte do geral para o particular. Assim, primeiro será definido o sistema, depois, serão definidos os componentes deste sistema, após os sub-componentes, etc. Cada um destes componentes deverá prestar um serviço de qualidade suficiente para que o composto venha a satisfazer o nível de qualidade esperado. Assim, ao identificarmos um componente devemos:

i) determinar a expectativa de qualidade do serviço deste componente; e

ii) verificar se esta expectativa é suficiente para assegurar o nível de qualidade esperado pelo composto que contém este componente.

Uma vez definida a qualidade do serviço, passamos a definir a qualidade do produto em si de forma que a qualidade do serviço possa vir a ser assegurada. Após, determinaremos o processo de desenvolvimento (plano, técnicas, métodos, etc.) que permitam desenvolver o produto de forma a garantir que a qualidade do produto seja assegurada.

Em um produto composto, são "usuários" de um componente os outros componentes com que este se relaciona. As "expectativas" destes "usuários" são as interfaces explícitas ou implícitas definidas entre estes componentes. Torna-se pois necessário definir precisamente estas interfaces, o que é conseguido através de definição de serviço de cada um destes componentes.

A avaliação da qualidade de um componente por parte de seu "usuário" ocorre quando da composição dos diversos componentes formando o produto composto desejado. A avaliação será satisfatória na medida em que esta composição se der sem causar dificuldades ou imprevistos, e na medida em que resulte num composto satisfazendo os requisitos de qualidade previamente estabelecidos. Esta inexistência de surpresas depende, é claro, da precisão com que a interface foi definida antes de se partir para a construção de cada um destes componentes. Depende, ainda, do rigor com que são examinadas e controladas as solicitações de alteração porventura necessárias nestas interfaces.

Uma vez de posse da especificação do serviço e da expectativa de qualidade do serviço dum componente, passamos a definir a qualidade do componente em si. Ao fazermos isto normalmente aparecem propriedades mais detalhadas. Por exemplo, do ponto de vista de um usuário convencional, a estrutura interna do programa em geral é irrelevante. Assim, ao definirmos o programa (especificação de requisitos) esta estrutura não deveria ser mencionada. Porém, no momento em que projetarmos o software para, por exemplo, possuir um grau elevado de alterabilidade, esta estrutura interna torna-se uma propriedade fundamental.

Através da hierarquização e detalhamento acima mencionado, poderemos manter sob controle o volume de propriedades a serem consideradas. Não somente isto, poderemos ainda adequar o nível esperado para cada uma destas propriedades ao nível efetivamente requerido. Por exemplo, em muitos sistemas é comum coexistirem sub-sistemas necessitando, por exemplo, elevada confiabilidade, com outros plenamente satisfatórios mesmo sem possuirem um grau elevado de confiabilidade. Especificando-se os níveis mínimos necessários e construindo-se o sistema para não ultrapassá-los de muito, aumentamos a economicidade do processo de desenvolvimento, garantindo o alcance de um nível de qualidade satisfatório.

(Ver figura 1).

Em resumo, a garantia de qualidade depende do processo de desenvolvimento. Para que possamos garantir a qualidade, este processo deve:

i) passar por diversos pontos de controle onde:

a) a qualidade alcançada pelos produtos desenvolvidos até o momento é comparada com a qualidade previamente definida (controle de qualidade);

b) se o produto concluído for uma especificação ou um projeto, avaliar se o resultado da construção caso feita em acordo com esta especificação poderá vir a ter o nível de qualidade esperado (predição da qualidade);

ii) seguir um plano de garantia de qualidade (IEEE81). Este plano estabelece, entre outros, os pontos de controle de qualidade, os procedimentos de controle, o cronograma destes pontos de controle, os resultados a serem examinados em cada ponto de controle, os procedimentos de controle de alteração de produtos já aceitos, os procedimentos de controle de configuração do software (Bersoff80, Gomes81),etc.;

iii) seguir uma metodologia de desenvolvimento de software estabelecida. Tal metodologia determina uma disciplina de trabalho assegurando padrões mínimos de qualidade e de visibilidade desta qualidade para todos os resultados e sub-produtos do desenvolvimento segundo esta metodologia. Foge ao escopo deste artigo examinar metodologias de desenvolvimento de software.

iv) definir (especificar) o problema a ser resolvido com rigor. É óbvio que um processo de construção excepcionalmente bem executado tendo por base uma especificação ruim, pode resultar no máximo num sistema insatisfatório. Ou seja, sem definições adequadas corremos o risco de produzir uma excelente solução, para o problema errado.

4. MODELO DE DEFINIÇÃO E AVALIAÇÃO DA QUALIDADE

Nesta seção delinearemos um modelo para a definição e controle da qualidade dum sistema. Por razão de escopo e de espaço, descreveremos o modelo com pouco detalhe. O leitor interessado em mais

detalhes deverá consultar as referências (McCall77, McCall79, Peercy81, Mazzoni81, Barros82).

O modelo utilizado é hierárquico e baseia-se em cinco conceitos fundamentais:

i) objetivos de qualidade – são as propriedades gerais que o software deverá possuir;

ii) fatores de qualidade do serviço – são atributos, condições ou características determinantes da qualidade do software do ponto de vista do usuário;

iii) fatores de qualidade de engenharia – são atributos, condições ou características da construção do software determinantes da qualidade deste software;

iv) métricas -- são medidas ou avaliações quantificáveis dos atributos de qualidade do software;

v) funções de normalização – permitem quantificar os diversos fatores em função de avaliações de métricas, tornando mais objetiva a discussão em torno da qualidade.

Ao definir-se a qualidade que um determinado software, componente ou sub-produto, deverá vir a ter, parte-se do geral para o detalhe. Definem-se em primeiro lugar os objetivos de qualidade, e estabelece-se uma relação de prioridade entre eles. Após, definem-se os fatores de qualidade do serviço e os pesos relativos de cada um destes fatores com relação aos objetivos de qualidade. Após, são determinadas as tolerâncias dos fatores, determinando:

i) limite de tolerância inferior – caso a avaliação do fator caia abaixo deste limite, o produto deverá ser revisto, mesmo que a avaliação de outros fatores tenha resultado num grau elevado;

ii) faixa condicional – faixa de graus de avaliação do fator permitindo a aceitação condicional com relação ao fator em questão, desde que outros fatores relacionados satisfaçam um grau de avaliação acima dum limite inferior especificado;

iii) limite satisfatório – caso a avaliação do fator caia acima deste limite, o produto poderá ser aceito com relação a este fator.

Após termos definido os fatores de qualidade do serviço, estabelecemos vínculos entre estes e os fatores de qualidade de engenharia. Para tal definimos funções de normalização permitindo a avaliação de fatores de qualidade de serviço em função de fatores de qualidade de engenharia. As tolerâncias dos fatores de qualidade de engenharia são estabelecidos em função dos objetivos de qualidade e das funções de normalização definidas.

As diversas funções de normalização devem produzir resultados entre 0 e 1, onde 0 corresponde à inexistência do fator, e 1 corresponde à satisfação integral do fator. Utilizando esta convenção, torna-se possível a definição dos níveis de aceitabilidade antes de sabermos como o fator será avaliado. As fórmulas utilizadas pelas diversas funções de normalização dependem do grau de relacionamento entre os diversos componentes destas fórmulas, variando caso a caso. Na maioria das vezes, as fórmulas serão médias ponderadas.

Uma vez definidos todos os fatores, passamos a definir as métricas. A noção de métrica contém em si a noção de dimensão (unidade de medida) e do processo de medição (avaliação) a ser utilizado. Ao definirmos uma métrica devemos, portanto, definir o significado da métrica, a "escala" (valores possíveis e a ordenação existente entre estes valores), bem como devemos definir como será efetuada a medição ou avaliação. Após termos definido as métricas, passamos a definir as funções de normalização que permitem a avaliação dos diversos fatores em função dos valores obtidos ao avaliar ou medir cada uma das métricas relacionadas com estes fatores.

Também no caso de métricas, as tolerâncias serão determinadas em função dos objetivos de qualidade e das funções de normalização já definidas. A determinação das tolerâncias de métricas assegura maior objetividade no desenvolvimento do software, uma vez que os diversos participantes saberão agora quais serão os critérios de avaliação específicos a serem utilizados para controlar os resultados de seus trabalhos.

A título de ilustração apresentaremos a seguir, exemplo de um modelo de avaliação de qualidade, alertando desde já para o fato do modelo estar incompleto por razões de escopo deste texto.

Os objetivos de qualidade são:

Utilidade – avalia a confiabilidade e utilidade dos resultados, bem como a tempestividade de sua geração;

Utilizabilidade – avalia a facilidade de uso do software nas diversas condições de contorno, bem como segundo os pontos de vista de seus diversos usuários;

Monitorabilidade – avalia a disponibilidade de instrumentos para o acompanhamento, avaliação e controle do comportamento do software e do serviço por ele prestado;

Evolutibilidade – avalia a capacidade de adaptação e alteração do software, visando adequá-lo a novas condições ambientais e/ou de serviço (manutenção);

Rentabilidade – avalia a economicidade no consumo dos diversos recursos necessários para obter o serviço do produto, bem como para adquirir o software em si.

Os objetivos de qualidade podem ser caracterizados pelas perguntas que deverão ser capazes de responder. Estas perguntas indicam, em linhas gerais, quais os fatores que compõem o objetivo de qualidade. Assim:

Utilidade deverá responder, entre outros, a:

– o software faz o que o usuário necessita e/ou espera?

– o faz sempre (determinismo)?

– o faz quando solicitado (disponibilidade)?

– os resultados aparecem quando necessários (tempestividade)?

– os resultados são confiáveis, mesmo em condições ambientais ou de uso adversas (confiabilidade)?

– protege-se contra condições ambientais e de uso adversas (robustez)?

– é capaz de continuar operando, pelo menos parcialmente, em condições ambientais e de uso adversas, sem produzir danos (ductibilidade)?

Utilizabilidade deverá responder, entre outros, a:

– sei operar?

– sei interagir com o sistema?

– sei como fornecer corretamente os dados necessários?

– sei quando fornecer os dados necessários?

– sei interpretar os resultados e mensagens fornecidas?

– sei para que servem os resultados e as mensagens?

– sei instalar?

– sei prevenir a ocorrência de acidentes, falhas, defeitos, erros de uso, erros de dados, etc.?

– sei como recuperar quando da ocorrência de acidentes, falhas, defeitos, erros de uso, erros de dados, etc?

– sei como medir, auditar, avaliar, controlar e monitorar?

– sei corrigir defeitos remanescentes?

Monitorabilidade deverá responder, entre outros, a:

– posso medir o consumo de recursos por componente?

– posso medir o contínuo alcance dos requisitos estabelecidos?

– posso controlar a qualidade do serviço prestado?

– posso auditar?

– posso examinar o efeito de dados e/ou condições selecionadas sobre o programa (sistema), durante o processamento?

– posso determinar a ocorrência de acidentes, falhas, defeitos, erros de uso, erros de dados, etc?

– posso determinar estar próximo a esgotar os limites de capacidade de recursos (memória auxiliar, tempo, etc.)?

(Ver figura 2).

Evolutibilidade deverá responder, entre outros, a:

– poderei mudar de equipamento?

– poderei mudar de software básico, suporte, etc?

– poderei adaptar a novas condições ambientais?

– poderei adaptar a novos algoritmos, procedimentos de cálculo, agregação, ordenação, tabulação, etc.?

– poderei ampliar o número de funções disponíveis?

– poderei melhorar o desempenho?

– poderei ampliar a demanda de serviço?

– poderei ampliar os limites de capacidade de recursos necessários ao processamento?

Rentabilidade deverá responder, entre outros, a:

– os recursos (CPU, memória, memória auxiliar, papel, pessoal, dinheiro, etc.) são consumidos com parcimônia?

– a solução é a mais simples ainda atendendo aos demais requisitos de qualidade?

– o algoritmo escolhido é suficientemente eficiente para o fim a que se destina?

– o custo total é compatível com o benefício total?

custo total: desenvolvimento, operação, manutenção, recuperação, prejuízos decorrentes de erros e falhas, etc.
resultado total: redução de gastos, redução de capital, aumento da receita, benefícios sociais, etc.

Na tabela a seguir são apresentados fatores de qualidade do serviço que contribuem com o alcance do objetivo de qualidade "evolutibilidade" (Peercy81). A evolução do software está intimamente relacionada com a possibilidade de alterá-lo segura e economicamente. Por exemplo, a mudança de CPU, de periféricos, de sistema operacional, do software de suporte utilizado, etc. alteram as condições ambientais requeridas pelo programa para poder operar satisfatoriamente.

Inteligibilidade (INTL) – um produto possui o atributo inteligibilidade na medida em que seu propósito e sua organização são compreensíveis ao observador (inspetor de qualidade, usuário manutenedor).

Modificabilidade (MODF) – um produto possui o atributo modificabilidade na medida em que facilita a incorporação de alterações, uma vez que a natureza destas alterações tenha sido determinada.

Validabilidade (VALD) – um produto possui o atributo validabilidade na medida em que facilita a certificação, teste, verificação e validação completa das alterações incorporadas.

Portatilidade (PORT) – um produto possui o atributo portatilidade na medida inversa do esforço necessário para transferí-lo de uma condição ambiental para outra.

Para podermos alterar um software, obviamente precisamos compreendê-lo, o que implica na necessidade do fator inteligibilidade. A alteração em si é possível, desde que o software seja modificável. Uma vez o software ter sido modificado, temos que re-examiná-lo para que tenhamos a certeza que a modificação não reduziu a qualidade deste software. Se este exame requer um re-teste integral do software e não somente das porções afetadas pela modificação, é claro que a evolução é dificultada. Por esta razão o re-exame deve poder ser restrito somente às porções alteradas e ainda assim ser confiável e completo.

Na figura 3 mostramos um exemplo, de faixas de tolerância e pesos para a avaliação do grau de satisfação dos fatores de qualidade de serviço relacionados com o objetivo "evolutibilidade". A figura deve ser entendida somente a título de ilustração e não como uma definição sempre válida para as classes de software mencionadas. Na realidade os pesos e faixas de tolerância dependem mais da aplicação ou função do software do que da classe a que pertencem.

A tabela a seguir apresenta os fatores de qualidade de engenharia relacionados com o objetivo "evolutibilidade".

Modularidade (MOD) – um produto possui o atributo modularidade na medida em que foi particionado logicamente em componentes e/ou módulos.

descriptividade (DSC) – um produto possui o atributo descriptividade na medida em que contenha informação sobre os objetivos, hipóteses, restrições, entradas, saídas, processamento, componentes, estado de configuração, etc.

(Ver figura 3).

Uniformidade de notação (UNO) – um produto possui o atributo uniformidade de notação na medida em que foi construído utilizando notação, terminologia e simbologia uniformes.

Simplicidade (SMP) – um produto possui o atributo simplicidade na medida em que não utilize organização, linguagem e técnicas de implementação desnecessariamente complexas, refletindo o uso de conceitos simples e de construções básicas.

Alterabilidade (ALT) – um produto possui o atributo alterabilidade na medida em que facilite a alteração física de algoritmos, organizações de dados, capacidades de processamento e armazenamento, diagramas, projetos, etc.

Instrumentação (INS) – um produto possui o atributo instrumentação na medida em que possua suporte para a certificação, teste, verificação e validação.

Na tabela a seguir são apresentados as métricas relativas ao fator modularidade. As avaliações das métricas resultarão sempre em valores entre 0 e 10,

0 significando a pior e 10 significando a melhor situação possível relativa à propriedade em questão. Para reduzir a subjetividade das avaliações, são definidos estimadores característicos. Possíveis valores para os quais não são definidos estimadores podem ser utilizados para estabelecer uma gradação mais fina na avaliação. Por exemplo, no caso de "tamanho da interface". A função da normalização que transforma a avaliação das métricas em uma avaliação do fator modularidade é uma simples média ponderada das avaliações das métricas, normalizada para o intervalo 0 a 1.

Métrica	Definição	Valores
coesão	mede os inter-relacionamentos dos componentes internos ao produto	10 funcional 9 função composta 6 sequencial 5 comunicacional 2 temporal 1 lógico 0 coincidente
tamanho da interface	mede o número de elementos com nomes distintos que figuram na interface do produto	10 menos de 3 9 – > 3 8 – > 5 7 – > 7 6 – > 10 0 mais de 30
tipo de conexão	mede o modo como são estabelecidos os relacionamentos através das interfaces	10 parâmetro 7 global, 6 COMMON, EXTERNAL 4 referência, índice 2 pointer 0 superposição (EQUATE, REDEFINES)
semântica da conexão	mede o significado do relacionamento através das interfaces	10 dado puro 7 condição de retorno, exceção 3 condição de entrada 1 "go to" de saída do módulo 0 modificação de código (p.ex. ALTER em Cobol)
tempo de ligação	mede o instante quando é estabelecido o relacionamento através das interfaces	10 – execução 5 – compilação 0 – programação
clareza de ligação	mede a necessidade de conhecimento de aspectos internos ao produto para poder-se fazer perfeito uso destes ("hiding" (Parnas72))	10 basta conhecer a interface 7 é necessário conhecer-se o algoritmo empregado 4 é necessário conhecer-se a estrutura de dados empregada 0 é necessário conhecer-se o código

5. CONTROLE DE QUALIDADE

Na seção anterior vimos como definir um modelo de avaliação da qualidade de um produto. É função do controle de qualidade efetuar as medidas e avaliações, indicando ao final a aceitação ou rejeição do produto sendo controlado. É claro que, para podermos controlar, é necessário um padrão de comparação. Este padrão, como vimos na seção anterior, faz parte do modelo de avaliação – as faixas de tolerância.

Contrário à definição do modelo de avaliação, o controle de qualidade opera do detalhe para o geral. Ou seja, primeiro são avaliadas as métricas. Depois, através das funções de normalização, os fatores e os objetivos de qualidade. Rejeições poderão ocorrer em qualquer uma destas ocasiões, sempre que avaliações cairem abaixo dos limites tolerados.

Como deve ter sido observado pelo leitor, as medidas são frequentemente subjetivas. Torna-se assim necessário utilizar um processo de avaliação o menos tendencioso possível, ou seja, um processo onde sejam mínimos os efeitos de distribuição devido à subjetividade. A sugestão comum nestas ocasiões é a de promover a avaliação por diversas pessoas e, após, comparar os resultados através de uma análise de variança. Uma forte discrepância entre as diferentes avaliações é uma indicação ou de que o processo de avaliação foi executado de forma falha, ou de que o produto é ambíguo. Ambos estes casos são indesejáveis. Para reduzir a probabilidade de falha no processo de avaliação, é sugerido em (Peercy81), que, além de um adequado treinamento dos inspetores, também seja conduzido uma inspeção de calibração, onde os critérios utilizados pelos diferentes inspetores serão ajustados. Ponto fundamental, no entanto, é a inspeção de qualidade ser conduzida por uma equipe diferente de que desenvolveu o produto.

São exemplos de técnicas de controle de qualidade de software: inspeções ("walk throughs"), certificação, teste, simulação, etc. Foge ao escopo deste artigo descrever em detalhe estas e outras técnicas.

6. CONCLUSÃO

Aos olhos de muitos, o modelo apresentado pode parecer complexo e custoso. Tal não é necessariamente verdade. Usualmente os objetivos, fatores e métricas tenderão a ser os mesmos nos diversos projetos de desenvolvimento de sistemas. Mais ainda, é altamente provável que até as funções de normalização pouco variem com relação aos diferentes projetos de desenvolvimento. Assim o custo inicial da criação do modelo de previsão e avaliação de qualidade de software pode ser diluído sobre os diversos sistemas desenvolvidos sob o seu controle. O custo operacional esperado não é significativamente maior do que processos de controle de qualidade menos sistemáticos. Com base nesta argumentação é sugerida a confecção de "check lists" de objetivos, fatores e métricas, a padronização das funções de normalização, e a calibração destas funções com o decorrer do tempo e o acúmulo de experiência no uso do modelo.

Quanto aos benefícios, é evidente que um software de boa qualidade tenderá a ter uma maior rentabilidade do que um software funcionalmente semelhante, porém de baixa qualidade. É evidente, também, que, exagerando-se nos níveis de qualidade, a rentabilidade poderá vir a

ficar comprometida. Isto reforça a necessidade de definir-se a qualidade desejável de modo que não se venha a ter prejuízos decorrentes tanto da falta como do excesso de qualidade.

BIBLIOGRAFIA

(ACM81) ACM/SIGMETRICS
1981 ACM Workshop/Symposium on Measurement and Evaluation of Software Quality; ACM/ SIGMETRICS 10(1); 1981.

(BARROS82) Barros, S.C.
Uma Metodologia para a Garantia, Definição e Controle de Qualidade de Sistemas Automatizados; Dissertação de Mestrado, Informática, PUC/RJ; 1982.

(BERSOFF80) Bersoff, E.H.; Henderson, V.P.; Siegel, S.G. – Software Configuration Management; An Investment in Product Integrity; Prentice-Hall, Englewood Cliffs, N.J.; 1980.

(BOEHM75) Boehm, B.W.
"The High Cost of Software;" em (Horowitz75); 1975; págs. 3-14.

(BOEHM78) Boehm, B.W. et al.
Characteristics of Software Quality; North Holland, TRW Series, New York; 1978.

(BROOKS75) Brooks, F.P.
The Mythical Man Month; Addison Wesley Pub. Co., Reading, Mass.; 1975.

(CHO81) Cho, C.K.
An Introduction to Software Quality Control; John Wiley and Sons, New York, 1981.

(COOPER79) Cooper, J.D.; Fischer, M.J. eds.
Software Quality Management; Petrocelli Charter, New York; 1979.

(GILB76) Gilb, T.
Software Metrics; Winthrop Pub. Cambridge, Mass.; 1976.

(GOMES81) Gomes, E.L.
Gerência de Configuração de Software; Dissertação de Mestrado, Informática, PUC/RJ, 1981.

(HOROWITZ75) Horowitz, E. ed.
Practical Strategies for Developing Large Software Systems; Addison Wesley Pub. Co., Reading, Mass.; 1975.

(IEEE81) IEEE Project P370
Draft Standard for Software Quality Assurance Plans; março 1981.

(MAZZONI81) Mazzoni, C.J.
Um Modelo para a Especificação e Avaliação da Qualidade de Software; Dissertação de Mestrado, Informática, PUC/RJ, 1981.

(McCALL79) McCall, J.A.
"An Introduction to Software Quality Metrics"; em (Cooper79); 1979; págs. 127-142.

(McCALL77) McCall, J.A.; Richards, P.K.; Walters, C.F. – Factors in Software Quality; RADC-TR-77-369; Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Basis, New York, 1977 (3 volumes).

(MOHANTY79) Mohanty, S.N. – "Models and Measurements for Quality Assessment of Software"; Computing Surveys ACM 11(3); setembro 1979; págs. 251.276.

(PARNAS72) Parnas, D.L. – "On the Criteria to be used in Decomposing Systems into Modules"; Communications ACM 15(12); dezembro 1972; págs. 1053-1058.

(STAA79) Staa, A.v. – Desenvolvimento Estruturado de Sistemas Automatizados; Monografia 9/79, Informática, PUC/RJ, 1979.

(STAA82) Staa, A.v.
Engenharia de Programas; Terceira Escola de Computação, Informática, PUC/RJ, 1982.

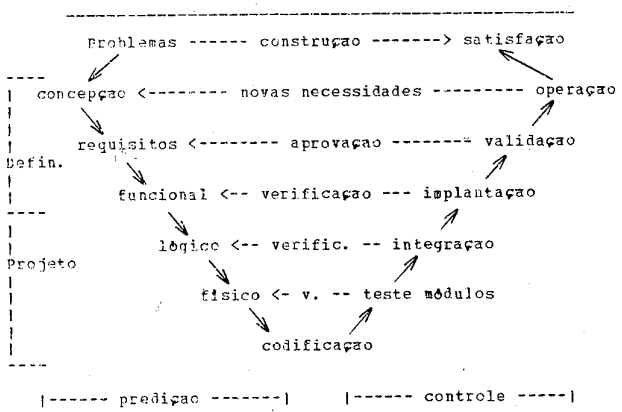

Figura 1. Predição e controle de qualidade vinculados ao ciclo de vida do sistema de software.

	utili	utilizab.	manut.	evol.	reutab.
aplicações comerciais convencionais, batch	1	4	3	2	5
aplicações comerciais convencionais, on line	1	2	3	4	5
controle de processo	1	4	2	5	3
demonstração	3	1	4	5	2
software básico	1	5	2	4	3
software suporte	1	3	2	4	5
processadores de linguagem (produção)	1	2	5	3	4
processadores de linguagem (teste)	1	3	5	4	2
software descartável	1	2	3	5	4

Figura 2. Prioridades genéricas dos objetivos de qualidade de algumas classes de programas.

	INTL	MODP	VALD	PORT
software produto	5 .7-.8	2 .5-.8	3 .6-.7	5 .7-.8
software sob medida	4 .7-.8	3 .6-.7	2 .6-.8	1 .5-.6
software descartável	2 .5-.7	1 .4-.5	1 .5-.6	0 .2

legenda: o número superior é o peso do fator. os pares inferiores determinam a faixa de tolerância mínimo-satisfatório.

Figura 3. Limites de tolerância dos fatores de qualidade do serviço do objetivo "evolutibilidade".

ENGRH. / SERVIC.	MOD	DSC	CNS	SMP	FLX	INS	lim. inf.
inteligibilidade	2	3	2	1	0	0	.8
modificabilidade	3	1	2	1	4	1	.8
validabilidade	3	3	2	2	1	4	.8
portatilidade	3	1	1	2	4	1	.6
limite inferior	.8	.7	.8	.7	.5	.6	

Figura 4. Funções de normalização de fatores de qualidade de engenharia para fatores de qualidade de serviço.