

QUALIMETRIA E GESTÃO DE QUALIDADE EM TI

por: André Balparda

Artigo apresentado durante o encontro anual do CMG-Brasil 2008
28 de agosto de 2008

Este artigo propõe uma investigação sobre o que é o termo Qualimetria, sua utilidade e aplicação em softwares. Bem como uma idéia sobre como a Qualimetria aliada a uma gestão de TI pode reduzir custos e aumentar a satisfação dos clientes.

1. Introdução

Este artigo não tem a intenção de ser um tratado final sobre o tema proposto, mas sim trazer à tona uma discussão sobre o que é Qualimetria e como esta pode ser útil para a realidade de desenvolvedores, gestores e trabalhadores da área de TI em geral, sem contar os enormes benefícios financeiros que resultam de sua utilização.

Qualimetria é um termo novo. Analisado morfológicamente, quer dizer: quantificação da qualidade. Em outras palavras, medir a qualidade de uma coisa, seja esta um carro, um celular, um martelo ou um software.

É obvio que o que iremos discutir neste texto é a capacidade de se medir qualidades intangíveis de um software, e como utilizar estas informações para uma melhor concepção, desenvolvimento e manutenção destes. Para atingir este objetivo, deveremos passar por algumas fases, partir da análise de uma coisa palpável e real, aprender como entendemos esta coisa e quais suas funções, para então entender um software, que não tem existência palpável, apenas virtual.

Este conceito, que por enquanto deve nos satisfazer, será vital para o bom entendimento deste artigo. Afinal, há muitas coisas que já são difíceis de se medir e se comparar em um mundo palpável, e isso é ainda mais complexo e difícil em um mundo virtual.

De alguma forma ou outra, já utilizamos Qualimetria em nosso dia-a-dia, seja na vida cotidiana, seja na análise de software, apenas não tínhamos colocado tudo isso em tabulação organizada e categorizada para que pudéssemos tirar proveito dessas informações de maneira metodológica e certa.

2. O que é Qualidade

Antes de entrarmos na Qualidade de Software propriamente dita, é essencial que entendamos o que é Qualidade, e como apreendemos sobre um objeto de estudo. Este passo nos guiará a percebermos o software de maneira diferente, tentando fugir da dicotomia *função versus performance*.

Ao consultarmos o dicionário, este nos ensinará que Qualidade é um predicado, ou disposição

moral, que diferencia uma coisa de outra. Em outras palavras, qualidade é aquilo que pertence a uma coisa e que faz com que esta, mesmo que essencialmente ela seja a mesma que outra, seja diferente. Um exemplo com certeza pode ajudar neste entendimento: uma cadeira qualquer é essencialmente igual a qualquer outra cadeira no mundo, e até mesmo uma cadeira ainda não existente fora da mente de seu construtor. Afinal, uma cadeira tem apenas uma função essencial: é o aparato no qual nos sentamos. Uma cadeira pode ser alta e outra baixa, uma amarela e outra verde, uma de madeira e outra de alumínio, e assim infinitamente, chegando ao extremo que duas cadeiras aparentemente idênticas podem ter sido fabricadas em anos diferentes, o que as torna diferentes quanto a durabilidade, ou, se quisermos ser preciosistas, duas cadeiras de plástico idênticas, do mesmo minuto de fabricação, terão plásticos diferentes. Afinal, o plástico que está em uma não está em outra.

Este preciosismo é fundamental para entendermos o que é a essência de uma coisa. No caso de um software, a sua função essencial. Entendendo isso, garantimos que esta função essencial nunca pode se alterar, independente de suas qualidades, esta função é que define o que o software é. Sendo assim, a princípio, se o software cumpre com sua função essencial, de certa maneira ele satisfaz sua concepção. Mas a que custo? O cliente deste software terá sua expectativa satisfeita? Foi esta função essencial que o cliente queria?

Essas qualidades das coisas não definem o que a coisa é, mas definem como interagimos com ela. Afinal, uma cadeira pode ser muito alta para uma pessoa, e esta mesma cadeira será baixa para outra. Isso é que caracteriza a expectativa do cliente ao produto. Se o cliente só ficasse satisfeito com a função essencial de uma coisa, todos nós dirigiríamos sempre o mesmo modelo e cor de carro. A pergunta que devemos nos fazer então é: o quão amarelo deve ser uma cadeira para satisfazer a expectativa do cliente? O quão alta deve ser? O quão extensa?

Antes de entrarmos nestes detalhes já ligados à Qualimetria, vamos tomar emprestados os conceitos de entendimento de Aristóteles sobre as coisas (Órganon 1b25). Aristóteles acreditava que todas as coisas podem ser explicadas por dez categorias, ou

seja, todas as predicções possíveis de um objeto de estudo devem se encaixar em algumas destas categorias a seguir: *Substância*, entidade única que é o sujeito de todas as outras categorias; *Quantidade*, pode ser discreta ou contínua; discreta quando as partes não possuem limites comuns (ex: 2 funções); contínua quando existem limites (20 Kb ou 5 metros); *Qualidade*: são as disposições, calor, frio, bondade; *Relação*: diz-se da relação quando existe dependência (isso é o dobro daquilo, então daquilo tem que existir); *Lugar*: onde a substância está (Paris, Têrreo, Servidor 5); *Tempo*, ordem relativa: Hoje, Ontem; ou duração: 5 segundos, 2 semanas; *Posição*: como se encontra a substância: deitado, sentado; *Estado*: ligado, armado; *Ação*: o que a substância faz: corta, queima, roda, executa, grava; *Paixão*: o que a substância sofre: cortado, queimado, alterado, gravado.

Esta maneira de se classificar os predicados de uma coisa não são mandatárias para software. Afinal, não foi feita para se categorizar um software, que tem peculiaridades distintas dos objetos reais e que, por exemplo, não possui volume espacial e forma. O que é mais importante no exemplo é a preocupação em tentar entender o objeto da melhor maneira possível. Este desafio ainda é válido hoje em dia, afinal, para podermos afetar de forma positiva a maneira com que as coisas interagem com outras é necessário alterar seus predicados e, talvez, sua própria essência. Assim, podemos entender o objeto de estudo, mas ainda não abordamos como entender a expectativa do cliente.

Uma maneira de se entender a Qualidade é a partir da falta dela. E muito mais fácil para nós entendermos o que não nos satisfaz daquilo que nos satisfaz. Este foi o ensinamento de Genichi Taguchi, guru da qualidade mundial, que disse: "Falta de Qualidade está diretamente relacionada à 'perda que um produto impõe à sociedade após sua distribuição'" e em sua teoria da Função de Perdas, que a muito grosso modo é uma relação matemática entre satisfação do cliente e especificação do produto.

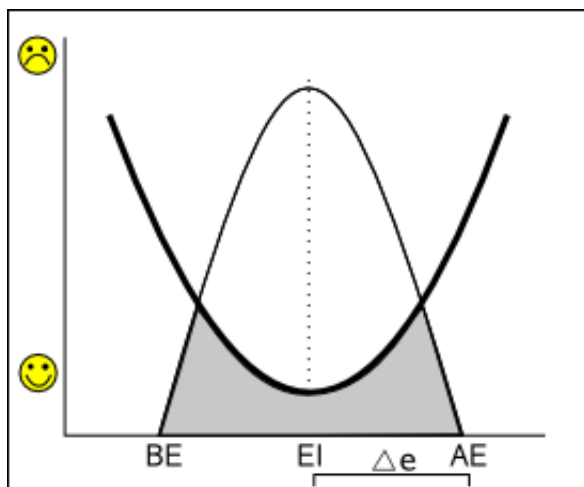


fig. 1 - Função de Perda (Genichi Taguchi)

A figura 1 ilustra muito bem a relação entre *satisfação do cliente versus a especificação do produto*. Onde o eixo horizontal indica o quanto se especifica um produto, sendo BE = baixa especificação, AE = alta especificação e EI = especificação ideal, a área cinza representa a perda com o produto. A melhor maneira de interpretar esse gráfico é, quanto menor o Δe (a distância que sua especificação está da especificação ideal), menos perda seu produto sofrerá.

Esta idéia está com certeza mais próxima à realidade da Tecnologia da Informação, pois a capacidade e necessidade dos sistemas mudarem é muito rápida, e em um tempo muito curto, algo que era decorativo passa a ser necessário da noite para o dia, tornando-se parte da especificação inicial.

3. Qualidade de Software

Quanto à qualidade de um software: o que podemos dizer de imediato? O que podemos esperar, antes mesmo de ver o software ser executado? Quais são nossas expectativas? Nem precisamos ser da área de TI para responder a essas perguntas, pois a resposta é muito simples: primeiro, o software precisa cumprir ao que foi concebido, ou seja, não pode ocorrer erro; segundo, deve ser executado rápido e com baixo custo de operação. Note que ainda não fugimos da dicotomia mencionada acima, só para lembrar: *função versus performance*. É obvio que existem vários outros fatores que qualificam um software, como: tamanho, plataforma para a qual foi desenvolvido, qual base de dados se utiliza, entre outras. Mas estas qualificações são independentes de sua função essencial.

Antes de entrarmos nos conceitos mais técnicos, é importante atentarmos para a expectativa do cliente. O que o cliente espera de um software ao encomendá-lo ou comprá-lo? Como mencionado acima, o cliente quer que o software funcione e que funcione rápido. Além disso, que seja fácil de usar, fácil de instalar, que não lhe dê manutenção e que seja o mais barato possível. Esta é a expectativa do cliente. Por isso, temos que ser muito preciosistas na especificação e lembrar que devemos especificar o necessário para cumprir com eficiência sua função. Se especificamos demais, demoramos para entregar o software e ele fica muito caro. Se especificamos de menos, não atende às necessidades do cliente. Este equilíbrio é algo muito delicado de se estabelecer.

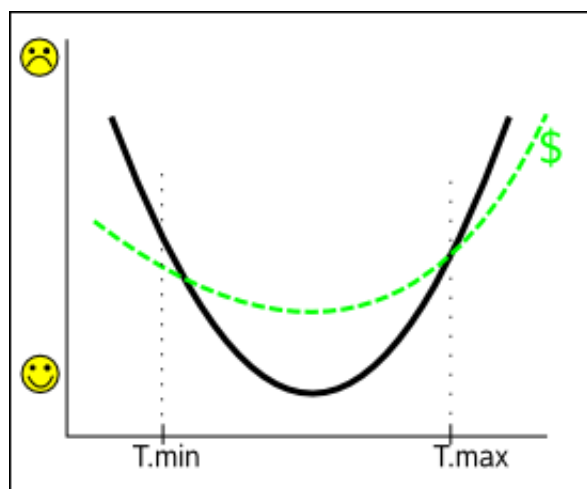


fig. 2 - Custos de produção

A figura acima ilustra no eixo vertical a insatisfação do cliente e no eixo horizontal o grau de especificação do software. T.min indica o ponto de tolerância mínima e T.max o de tolerância máxima do cliente. A linha pontilhada representa o investimento no projeto, o seu custo de fabricação. Sendo que um software mal especificado levará a erros de projeto e retrabalho aumentando então o seu custo, uma especificação excessiva tem como consequência um longo “time-to-market” e alto custo de desenvolvimento. Achar o ponto de especificação ideal é a melhor maneira de manter baixo custo de projeto e expectativa do cliente satisfeita.

Mas como medir? Como quantificar esta expectativa? Mais importante ainda, como quantificar a satisfação desta expectativa? Obviamente, não é possível dizer que a satisfação é um valor, mas, comparativamente, podemos fazer uma escala. Então, podemos pontuar, dar notas à nossa satisfação, por exemplo: “Este software ter nota 'B'”. Essa comparação depende da experiência, temos que já ter usado algum software similar para ter esta opinião. Isso não compromete a pesquisa quando falamos de um software comum, ou algo que todos estão acostumados naquele meio, pois todos já têm certeza (ou pelo menos acham que têm) do que esperam daquele produto. Neste caso, tabulamos todas as opiniões e obtemos uma nota média de satisfação. Mas ainda não temos qual é a expectativa, pois elas sempre serão as mesmas, *tem que funcionar e funcionar rápido*. É muito difícil interpretarmos o que o cliente quer, por ser tão subjetivo. É por este motivo que gastamos, como profissionais de TI, tanto tempo e dinheiro em prototipagem, fluxogramas, UMLs, homologações funcionais, reuniões e reuniões para aprovarmos um projeto. Tanto o cliente quanto o fornecedor sabem que esta fase de entendimento é complexa e delicada, por isso existe uma série de metodologias disponíveis para garantir o mínimo de verossimilhança entre escopo e software.

Todas essas práticas, metodologias e recomendações (CMM, ITIL, ISO 9000, COBIT, ISACA, IEE-829 e outras) mostram o que fazer, mas não garantem os processos de qualidade. Historicamente, várias empresas com altos índices obtidos em certificações já ofereceram softwares com problemas gravíssimos. Ou seja, todo este trabalho não dá conta do que o cliente espera. Isto porque nenhuma destas metodologia citadas automatiza o processo de qualidade.

Automatizar o processo de qualidade de software significa que nenhum software poderia ser compilado, sem antes ser “aprovado” por um *engine* que compara seu código atual com uma série de regras qualimétricas, trancando sua entrada em produção caso não esteja de acordo em 100% com as metodologias, práticas e recomendações citadas acima.

Além disso, novas tecnologias e métodos surgem a cada instante, isto muda o ambiente em que os softwares estão, muda a maneira como eles devem interagir com outros softwares e com o usuário. Contemplar estas mudanças é a missão mais difícil ao se conceber um software. Temos que ao menos garantir que naquele ambiente para o qual o software foi desenvolvido nada de errado aconteça. Para que isso aconteça, precisamos de dados qualimétricos e de gestão.

4. Qualimetria

Se utilizarmos a idéia do Sr. Genichi Taguchi para analisar a qualidade de um software, chegaremos à seguinte conclusão: a ausência de qualidade em um software se manifesta por meio de erros (funcionais e *abends*), perdas à corporação (sejam elas jurídicas, financeiras ou de imagem), fraudes, perda de competitividade, alto custo e grande “time-to-market”. A origem destas “falhas” na qualidade podem ser várias, entre elas: falta de padronização, descumprimento às boas práticas de programação, retrabalho, falta de testes, impacto crescente nos custos de produção, excesso de terceirização e alta complexidade dos sistemas.

Começamos então a entender qual é o desafio da Qualimetria de software. Infelizmente, ainda não conseguimos nos desfazer totalmente daquela dicotomia *função versus performance*, mas adicionamos uma nova variável nessa equação: o **custo** (fig. 2 também exemplifica isso). Afinal, desenvolver um software que seja “state of the art” e ainda supere todas as expectativas do cliente pode ser possível. Mas será economicamente viável? Será entregue no prazo? Mais importante, saberemos como desenvolver isso e como anteciparemos problemas em produção? Esta resposta não está muito longe, pois já temos algumas ferramentas capazes de analisar código e buscar por regras pré-definidas, evitando que ele vá prematuro para a produção.

O segredo da Qualimetria de software é muito simples; o primeiro passo é obter, armazenar e categorizar as informações obtidas em produção e as informações das expectativas do cliente; o segundo passo é transformar todas essas informações em ações, que durante a concepção de um novo software saberão prevenir todas as falhas já mencionadas acima, antes mesmo da existência do novo software.

Quantificamos então a expectativa do cliente por meio de notas e modelos comparativos. Mas se formos especular sobre um produto ou software jamais feito ou imaginado, com certeza esse modelo comparativo terá suas falhas, pois não teremos algo já criado para compararmos, então, nossa expectativa será baixa. Mas, assim que este software entra em produção, opiniões começam a ser formadas e comparações poderão ser feitas. Este histórico de ampliação de expectativas deve ser também armazenado e tabulado para futuras ações.

Como podemos então nos preparar para transformar informações de produção em ações ao iniciarmos um novo projeto? Não foi em vão que acima falamos das categorias aristotélicas, esta metodologia, se é que podemos chamá-la de metodologia, pode servir de guia para classificarmos informações sobre execução e erros de um software. Obviamente, ela não é ainda o ideal, mas já é uma base inicial para aprendermos a entender o software de maneira mais completa. Obter o histórico de execução é a chave para podermos atingir nosso objetivo, é por meio destas informações que criaremos regras que orientarão concepções de novos softwares. Classificar estas informações é útil para poder ajudar a transformá-las em ações, ou regras. Devemos organizar toda essa massa de dados estatísticos de maneira a classificá-la de acordo com ambiente, local, rede, dependências, tamanho, prioridade, tempo de execução, sistemas atingidos, volume de transações, autoridades, e tudo mais que interage com este software.

Parece claro tudo isso que falamos até então, mas nem sempre amarramos informações como dependência entre softwares com erro do software, a não ser que seja relativo ao próprio código de erro. Uma visão completa é o objetivo dessa massa de dados, não só armazenarmos os erros ocorridos, mas também os acertos. É a capacidade de sabermos em quanto tempo “rodou” um programa que tem dependência com outro em um dia chuvoso de março. Este tipo de informação super detalhada serve para podermos, como dito acima, diferenciar o plástico de cada cadeira, afinal o plástico que está em uma, não está em outra, ou em idioma de TI, a alocação de memória utilizada em uma execução, não é a mesma alocada em outra.

Organizar, categorizar e armazenar toda essa massa de dados já é um desafio bastante grande. Mais interessante ainda, é como transformar isso em regras de concepção e de codificação. Tudo mencionado até então, pode ser feito por “engines”, softwares previamente instruídos para tal. O desafio é como

transformar esta massa de dados de informação em regras de Qualimetria, como interpretar estas informações e transformá-las em conhecimento.

Felizmente, máquinas e sistemas ainda não conseguem executar este salto entre a informação e o conhecimento, isso ainda é papel do homem. A capacidade de interpretação de detalhes sutis e a capacidade de unir duas idéias diferentes é que torna o homem essencial neste processo. A máquina ainda não é capaz de entender o subjetivo, e Qualimetria é exatamente isso, quantificar algo subjetivo.

Mas o que fazer com estas regras depois que foram criadas? Como garantir que serão seguidas à risca? Podemos lidar com isso de três maneiras diferentes, a saber: comportamental, operacional e física. Comportamental é quando na especificação, no projeto lógico ou no projeto físico contamos com a boa vontade dos participantes em seguir com as regras estabelecidas. Operacional é quando por meio de relatórios e listas de conferência fazemos com que todos os envolvidos em todas as fases da concepção do software se comprometam a cumprir com as regras. Lidamos de maneira física, quando no projeto físico só permitimos que um código passe para a fase seguinte se cumpridas todas as regras, ou seja, um código jamais poderá ser compilado se não seguir os parâmetros pré-definidos.

Contar com a boa vontade dos funcionários para seguir as boas práticas e regras com certeza dará margem para erros que podem comprometer a funcionalidade e performance dos softwares. Já uma abordagem física, não permitirá que o código seja compilado sem que esteja 100% de acordo com a tabela de regras e normas. Isso faz com que tenhamos certeza que nada será esquecido, economizando assim retrabalho, garantindo maior estabilidade e disponibilidade dos softwares. Em outras palavras, diminuir o retrabalho, padronizar código, testar com consistência de dados, prever o impacto no ambiente, diminuir os erros e “abends”, simplificar os programas e controlar as fábricas de software. Isso aumenta a competitividade de sua empresa e reduz seu “time-to-market”.

Se colocarmos estes dados em uma balança financeira, veremos que na atividade de desenvolvimento de software 40% do tempo é empregado em retrabalho (fonte: SEI – *System Engineer Institute*), o que significa que algo foi feito errado, ou não funcionou como esperado. Este tempo de retrabalho poderia ser empregado em trabalhos mais nobres como novos projetos ou evolução de produtos. Este tipo de impacto é brutal nos custos de qualquer empresa.

5. Qualimetria e gestão de TI

Ferramentas de qualidade de código, cruzadas com o repositório de informações de qualidade e relatórios de performance, tornam-se um grande aliado na gestão da TI. Podendo “pontuar” código de fábricas

de software, servir como meta anual para bônus dos desenvolvedores, ajudar a elaborar relatórios financeiros de impacto de crescimento. Relatórios de estabilidade e disponibilidade dos sistemas cruzada com número de transações pode também ser uma grande ferramenta em caso de termos de enfrentar uma fusão entre empresas. Todas essas diferentes visões (relatórios) deste repositório de informações, ajudam os profissionais envolvidos no processo de software, desde o desenvolvedor que pode entender melhor qual a utilização do software que vai desenvolver, para o usuário que pode saber qual é próximo passo de evolução deste software e entender sua participação nesta mudança, até o financeiro que poderá entender qual o impacto que uma mudança em um software, ou que um investimento em infraestrutura, terá sobre o custo de operação da empresa.

Nenhuma destas visões seria possível sem um procedimento qualimétrico instalado. A Qualimetria aliada à gestão de TI permite uma visão do histórico de eventos relacionada com as metas de qualidade que se pretende atingir, mostrando qual o impacto e até quando é que vale a pena se atualizar um programa legado da empresa.

Um bom exemplo é quando se tratando de fábricas de software, podemos estabelecer um mínimo de qualidade e por meio de um *engine* pontuar os pontos qualimétricos deste. Isso garante um padrão de qualidade aceitável do código, mantendo estabilidade e disponibilidade no ambiente da empresa. Sem mencionar a manutenibilidade do código que estará seguindo as boas práticas. Isso diminui os riscos de quebra do SLA e comprova os esforços no controle de qualidade, obtendo assim novos benefícios como: Controlar e aprimorar o SLA das fábricas de software; Controle e compatibilidade nos testes (garantindo que um teste manterá os mesmos padrões dos testes anteriores); Desempenho comparativo; Seleção dos melhores códigos e práticas; estabelecer metas para a manutenção e evolução dos softwares; Controlar *custo versus benefício*.

Quando utilizamos isso, sabemos que um software foi entregue pela fábrica de software, ou pelo departamento de desenvolvimento, com um fator qualimétrico de X e que foi implantado com X + Y após a padronização deste código, atingido assim os patamares mínimos de qualidade. Com o histórico deste software, podemos avaliar os padrões de qualidade dos fornecedores, medindo sua eficiência e podendo assim tomar uma decisão mais acertada.

Todos os testes e medições obtidos devem sempre alimentar o repositório de informação da gestão e da Qualimetria para manter um histórico evolutivo dos software, fornecedores, custos e ambiente. Assim, poderemos obter relatório que nos mostrarão os ganhos financeiros e qualimétricos obtidos com as regras implementadas.

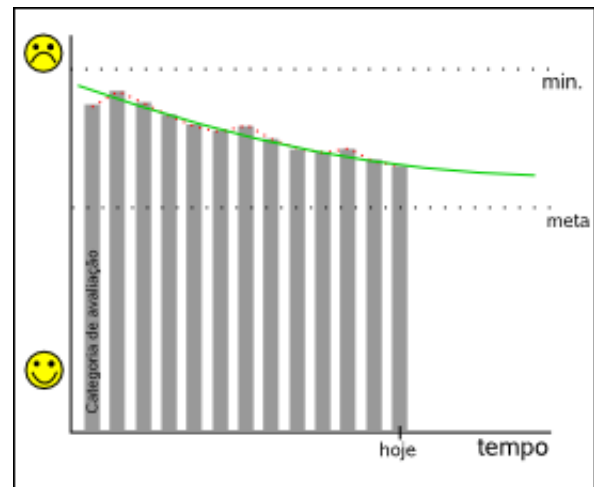


fig. 3 - Gestão de TI e qualimetria

Acima temos um exemplo de como podemos definir uma meta qualimétrica, baseada em alguma das categorias de qualidade (ex. *elapsed time*). A partir deste tipo de relatório, podemos projetar se, e quando, atingiremos a meta de qualidade estabelecida, e também não permitir que um software entre em produção sem que o mínimo de qualidade esteja garantida. Sabendo quando atingiremos essa meta, sabemos também quantas horas/homem serão gastas, assim, sabemos quanto custará o projeto e até quando ele é viável. Esta meta deve sempre respeitar o Δ explicado na figura 2, ela deve estar o mais próximo possível da especificação ideal, para que assim mantenhamos o equilíbrio *custo versus especificação*.

Todo este ambiente qualimétrico e de gestão permite uma comparação de aplicações legadas e novas. Dados qualimétricos obtidos pelos *engines* aplicados nos ambientes de desenvolvimento, homologação, testes e produção, permitem a criação de novas metas. Isso mantém as aplicações em sincronia com o crescimento vegetativo da empresa, garantindo que a curva de qualidade do software acompanhe a curva de crescimento da empresa. Em poucas palavras, garante estabilidade e disponibilidade.

Todos esses fatores de controle se tornam a maneira de garantir e automatizar as certificações de qualidade. Empresas gastam milhões de dólares para obter um certificado de garantia de seus produtos, mas acabam não prestando muita atenção em como manter este certificado. A Qualimetria, aliada a uma gestão de TI atuante, além de garantir estes certificados e baixo custo de fabricação, garante que seus softwares não possuirão erros potenciais futuros e que executarão de forma rápida e com baixo custo de produção.

Garantir disponibilidade, estabilidade, baixo custo, menor "time-to-market", códigos mais simples e de maior manutenibilidade, rápida evolução, menor retrabalho, maior desempenho, performance e satisfação das expectativas do cliente são o desafio definitivo de qualquer departamento de TI, e Qualimetria e gestão de TI são as ferramentas para superar este desafio.