GQM

Justificativas para medição:localizar gargalos no processo de desenvolvimento; identificar características desejáveis de uma classe de artefatos; identificar características a serem evitadas de uma classe de artefatos; identificar riscos reais; auxiliar a construir conjuntos de padrões, diretrizes e ferramentas úteis; avaliar o efeito das mudanças tecnológicas.

GQM: proposto como uma abordagem orientada a objetivos para a medição de produtos e processos, deriva métricas de *software* a partir de perguntas e objetivos.

Parte da premissa de que, para ganhar uma medida prática, deve-se antes entender e especificar os objetivos dos artefatos de *software* que estão sendo medidos, bem como os objetivos do processo de medição.

GOAL: quais são as metas/objetivos?

QUESTION: quais questões se deseja responder?

METRIC: quais métricas poderão ajudar?

Suas fases são: planejamento, definição, coleta de dados, interpretação.

Vantagens do GQM: apoia a definição *top-down* do processo de medição, e a análise *bottom-up* dos resultados; ajuda a identificar métricas úteis e relevantes; apóia a análise e interpretação dos dados coletados; permite avaliar a validade das conclusões tiradas; diminui a resistência das pessoas contra processos de medição. Outro aspecto positivo é sua aplicabilidade generalizada (não existem restrições quanto aos objetivos), a possibilidade de adaptar as métricas às características dos projetos ou das organizações, o apoio metodológico, o envolvimento de todos os interessados, a separação de papéis.

Fatores de sucesso técnicos: quais as métricas utilizadas, procedimentos de coleta de dados (não-intrusivo, disponibilidade de ferramentas de coleta e análise, independência de pessoas), qualidade das medidas (rigor da medição e da análise), treinamento, comunicação

dos resultados (feedback, observação da utilidade dos resultados pelos desenvolvedores e gerentes.)

Fatores de sucesso organizacionais: efetivo compromisso das pessoas envolvidas com o programa de medição; disponibilidade de recursos suficientes; apoio gerencial; maturidade da organização; crenças da organização.

Problemas do GQM: as métricas não são definidas no nível de detalhes necesśario para garantir confiabilidade, principalmente porque não é explicitado se uma métrica deve obter o mesmo resultado para duas avaliações diferentes realizadas por duas pessoas diferentes. Para melhorar um pouco este problema, pode-se categorizar as métricas em tamanho, esforço, planejamento, qualidade, desempenho, confiabilidade e complexidade. Então, para cada uma dessas categorias, propõe-se um conjunto de métricas que são agrupadas em classes de atributos relacionados ao *software*. Outro problema é não conseguir estipular quanto custará medir, armazenar e processar, nem os benefícios esperados, nem a viabilidade etc.: não se preocupa com os problemas relacionados com a medição em si.

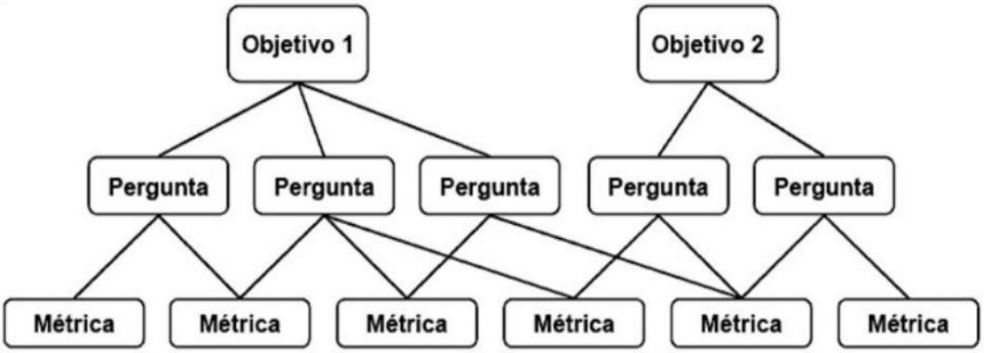
Fatos a se considerar no GQM: proteção de dados sensitivos (necessidade e o uso de dados é definido e acordado antes, e uma estrutura pode ser utilizada para criar mecanismos de controle de acesso) e aspectos éticos (o processo de medição, análise e divulgação deve ser previamente estabelecido, permitindo os envolvidos a se manifestarem quanto a possíveis invasões de privacidade ou divulgação de dados confidenciais.)

Passos básicos:

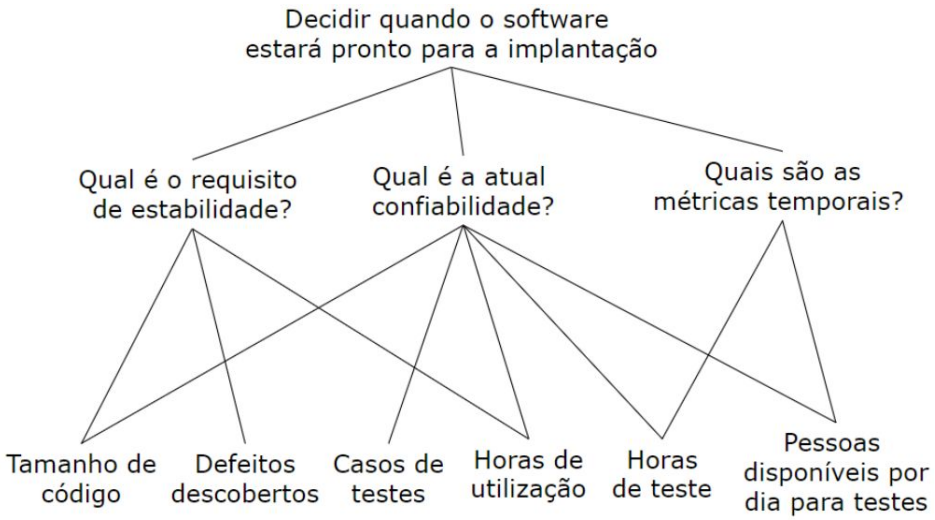
Nível conceitual (objetivos): listar os objetivos (em termos de entidade, propósito, atributos de qualidade, pontos de vista e ambiente) do processo de medição;

Nível operacional (conjunto de questões): refinar os objetivos em um conjunto de perguntas, que representam a definição operacional do objetivo, ou seja, para depois determinar se os objetivos foram atingidos;

Nível quantitativo (dados a apurar ou medir): para cada pergunta, definir as métricas (decidir o que precisa ser medido) capazes de responder às perguntas adequadamente.



Premissas para medição: prover resultados consistentes; permitir obtenção por não-especialistas em T.I.; ser fácil de aprender; ser compreensível para o usuário final; servir para estimativas; permitir automatização; possibilitar obter séries históricas.

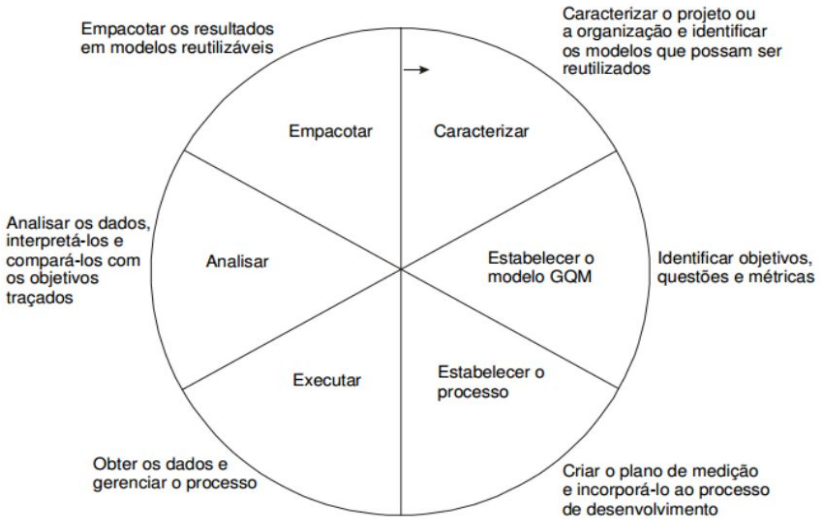


Etapas:

Desenvolvimento do Plano GQM: pré-estudo; elaboração do Plano GQM; elaboração do Plano de Avaliação.

Execução do Plano de Avaliação: coleta de dados a partir de formulários projetados na etapa de desenvolvimento do GQM e validação destes dados de acordo com os objetivos; tratamento dos dados por análise e técnicas de levantamento e estatística.

Preparação dos Resultados: preparação da documentação final; composição da base de experiências, com reutilização da base de conhecimento e com proposta de estratégias de melhoria da qualidade.



Identificação dos objetivos: analisar <objeto> com o propósito de <propósito> com relação ao seu (à sua) <foco> do ponto de vista de <perspectiva> e no contexto de(as, os) <contexto>.

Objeto	Propósito	Foco	Perspectiva	Contexto
produto	caracterizar	eficácia	desenvolvedor	pessoas
processo	monitorar	custo	mantenedor	objetos
modelo	avaliar	confiabilidade	gerente	linguagens
métrica	predizer	manutenibilidade	direção	tecnologia
teoria	controlar	portatilidade	cliente	projeto x
técnica	modificar	...idade	usuário	
			pesquisador	

MÉTRICAS E MEDIDAS

Sem medições e *feedbacks*, o controle é impossível, porque não se pode controlar o que não se pode medir. Portanto, a extensão do controle depende da precisão da medição, e qualquer coisa que não se pode medir está fora de controle. Afinal, medições e métricas ajudam a entender o processo técnico usado para desenvolver um produto e o próprio produto.

Métricas porém suscitam algumas polêmicas, como: quais métricas usar?, como os dados compilados devem ser usados?, é justo usar medições para se comparar pessoas, processos e produtos?

Há três pontos de vista, pelos quais o *software* deve sempre ser avaliado: o ponto de vista da operação do produto; de sua revisão, e de sua transição (migração de ambiente.)

Medir é importante para determinar melhorias.

Medições são importantes para determinar parâmetros como a quantidade de testes necessários, além do impacto de mudanças.

Medição: documentação de efeitos passados.

Uso: previsão de quantificação de efeitos futuros (estatísticas não podem prever, mas sim deduzir e, portanto, são utilizadas para projetar o desempenho futuro.)

Medidas subjetivas: baseadas em idéias individuais sobre o que deveria ser a métrica. O resultado será diferente com diferentes observadores.

Medidas objetivas: pode ser calculada precisamente, de acordo com um algoritmo. Seu não valor não se altera devido a alterações no tempo, local ou observador.

Estimativas: pode-se estimar o esforço humano exigido, a duração cronológica do projeto, o custo. Estimativas são a essência da dificuldade em controlar projetos de *software*.

Causas de estimativas mal-feitas: falta de treino; falta de se fazer provisões adequadas para contrabalancear o efeito das distorções; falta de conhecimento exato sobre o que é uma estimativa; falta de habilidade com os problemas políticos; falta de informações úteis para o processo de estimativas.

Atributos comuns de técnicas de estimativa: o escopo deve ser estabelecido antecipadamente; métricas são utilizadas; histórico de medidas passadas é usado como uma base; o projeto é particionado em pequenas partes, estimadas individualmente.

Análise de riscos: aspecto crucial para um bom gerenciamento de projeto de *software*; pode haver riscos técnicos, em se desviar do cronograma ou do custo etc. Riscos são áreas de incertezas. Um gerenciamento de projeto eficaz trata de identificar tais áreas e administrá-las adequadamente: avaliando os riscos; colocando-os em ordem de prioridade; estabelecendo estratégias de administração dos riscos; resolvendo os riscos; e, por fim, monitorando os riscos.

Determinação de prazos: envolve o planejamento da programação a ser realizada; a identificação do conjunto de tarefas de projeto; o estabelecimento das interdependências entre as tarefas; a estimativa do esforço associado a cada tarefa; a atribuição de pessoas e outros recursos para a realização e tarefas específicas.

Monitoramento e Controle: atividades que iniciam após o estabelecimento da programação de desenvolvimento, como: rastreamento no programa de desenvolvimento; determinação do impacto do não cumprimento dos prazos; redirecionamento de recursos, reorganização de tarefas, modificação nos compromissos de entrega.

Métricas de produto: servem para avaliar a qualidade do produto de *software*. Podem não revelar nada sobre como o *software* foi desenvolvido, e incluem o tamanho do produto (linhas de código), a complexidade da estrutura lógica (recursão, fluxo de controle, profundidade de laços aninhados), a complexidade da estrutura de dados, funções (*software* para ciências, *software* comerciais etc.)

Métricas de processo: servem para melhorar os processos. Quantificam atributos do processo de desenvolvimento e do ambiente de desenvolvimento. Métricas de recursos envolvem a experiência do programador; o custo de desenvolvimento e manutenção. Métricas

para o nível de experiência pessoal envolvem número de anos que uma equipe está usando uma linguagem de programação; número de anos que um programador está na organização etc.

Às vezes, é difícil classificar uma métrica entre de processo ou de produto. Por exemplo, o número de defeitos descobertos durante o teste formal depende do produto e do processo usado na fase de teste.

Medidas diretas do processo de engenharia de software: custo e esforço aplicados.

Medidas diretas do produto: número de linhas de código produzidas, velocidade de execução, tamanho de memória, número de defeitos registrados em um tempo especificado.

Medidas indiretas do produto: qualidade, funcionalidade, complexidade, eficiência, confiabilidade, manutenibilidade etc.

Métricas orientadas ao tamanho: são medidas diretas do *software* e do processo por meio do qual ele é desenvolvido. Existem várias maneiras de representar o tamanho ou magnitude de um programa, como a quantidade de memória necessária para armazenamento ou o número de linhas de código (qualquer linha do texto exceto comentários e linhas em branco, sem contar também o número de comandos ou fragmentos de comandos em uma linha, ou seja, são todas as linhas de cabeçalho, declarações e comandos executáveis.)

Vantagens de medir pelo número de linhas de código: fácil de calcular, e é o mais importante para muitos modelos de estimativa.

Desvantagens: dependência da linguagem de programação, o fato de que penalizam programas bem estruturados mas curtos, e o uso de estimativas requer um nível de detalhes que pode ser difícil de conseguir no início do projeto (é difícil determinar, afinal, quantas linhas o *software* terá.)

Métricas orientadas à função: são medidas indiretas do *software*, concentrando-se na funcionalidade ou utilidade do programa. Uma função é uma coleção de comandos executáveis que realizam certa tarefa.

Para calcular essa métrica, pode-se usar o Método do Ponto por Função de Albrecht. Para essa medida, cinco características do domínio da informação são consideradas, sendo elas os números de: entradas do usuário; saídas do usuário; consultas do usuário; arquivos (ou agrupamentos lógicos em um BD); interfaces externas.

Vantagens: independe da linguagem de programação; baseia-se em dados que têm mais chance de já serem conhecidos no início do projeto.

Desvantagens: baseada em dados subjetivos; dados do domínio da informação são difíceis de serem compilados a posteriori; não tem nenhum significado físico.

Métricas de qualidade podem ser aplicadas durante o processo de criação e após a entrega do *software*.

Métricas aplicadas durante o processo: base quantitativa para tomadas de decisão referentes a projeto e testes (complexidade do programa, modularidade.)

Métricas aplicadas após o processo: dão indicação pós-morte da efetividade do processo de engenharia de *software* (número de defeitos, manutenibilidade.)

Medidas de qualidade:

Corretude: grau em que o software executa a função que é dele exigida. Medido em defeitos/kloc, sendo defeito uma falta verificada de conformidade aos requisitos.

Manutenibilidade: facilidade com que um programa pode ser: corrigido se um erro for encontrado; adaptado se o ambiente for modificado; ampliado se o cliente desejar novas mudanças. É medida em tempo médio para mudanças (tempo para entender a mudança, para projetar uma alteração adequada, implementar a mudança, testá-la e colocá-la em operação.)

Integridade: mede a capacidade que um sistema tem de suportar ataques à sua integridade. Tais ataques podem ser feitos tanto a programas, dados, e documentos. Sua medida é $\Sigma(1-a * 1-s)$, sendo a a probabilidade de que um ataque de um tipo ocorrerá em um determinado tempo, e s a probabilidade de que tal ataque será repelido. Ambos valores são derivados de evidência empírica.

Usabilidade: tentativa de medir o quanto um programa é amigável ao usuário. Pode ser medida por: habilidade física ou intelectual para aprender a trabalhar com o sistema; aumento da produtividade sobre a abordagem que o sistema substitui; o tempo exigido para se tornar moderadamente eficiente no uso do sistema; avaliação subjetiva dos usuários em relação ao sistema.

VERIFICAÇÃO E VALIDAÇÃO

Objetiva assegurar que o *software* cumpra suas especificações, e atenda às necessidades dos usuários e clientes.

Ocorrem por todo ciclo de vida do *software* (há revisões de requisitos, de *design*, há testes de código.)

Verificação: o *software* deve estar de acordo com sua especificação. Analisa um sistema para certificar se este atende a requisitos funcionais e não-funcionais.

Validação: o software deve atender às necessidades dos usuários. É a certificação de que o sistema atende às necessidades e expectativas do cliente.

Deve-se verificar:

Fatores de Qualidade Operacionais: correção; eficiência ou desempenho; robustez; confiabilidade; usabilidade; utilidade; validade.

Fatores de Qualidade de Revisão: relacionados com a manutenção, evolução e avaliação do *software*.

Fatores de Qualidade de Transição: relacionados com a instalação, reutilização e interação com outros produtos.

Falta: oriunda de equívocos do desenvolvedor. Um equívoco pode causar várias faltas ao mesmo tempo, e vários enganos podem causar uma falta idêntica.

Falha: comportamento incorreto apresentado por um *software*, em consequência de uma falta.

Erro: representa o quanto um resultado é incorreto.

Defeito: termo genérico para falta, falha ou erro.

Inspeção (V&V estática): análise da documentação e do código fonte de software, pode ser auxiliada por ferramentas de documentação.

É uma técnica preventiva, permitindo V&V antes da codificação. É também mais barata. No entanto, costuma ser pouco eficaz para fatores operacionais, sendo mais aplicada a fatores de revisão e transição. É baseada na experiência do inspetor.

Suas aplicações mais comuns incluem a inspeção de programa-fonte, de documentos e de modelos.

Testes podem ser aplicados em fatores operacionais tais quais: correção; usabilidade; desempenho; robustez.

Testes (V&V dinâmica): o programa ou protótipo podem ser executados. Casos de testes devem ser elaborados a partir de dados de entrada e de comportamentos esperados.

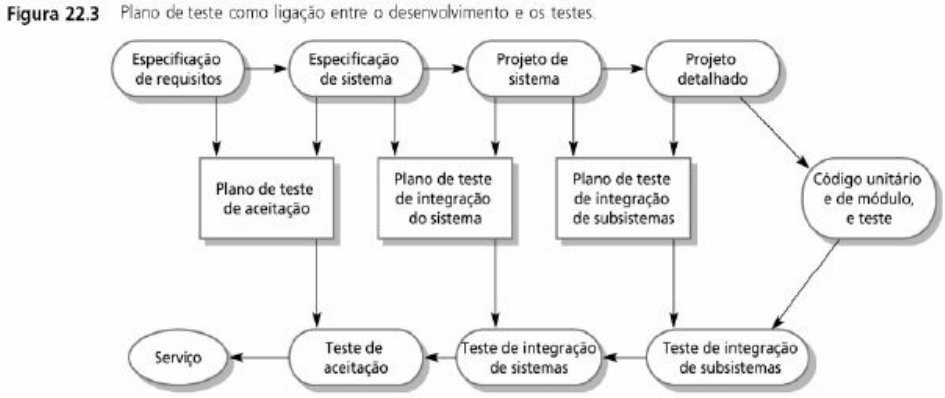
Testes podem ser classificados quanto ao:

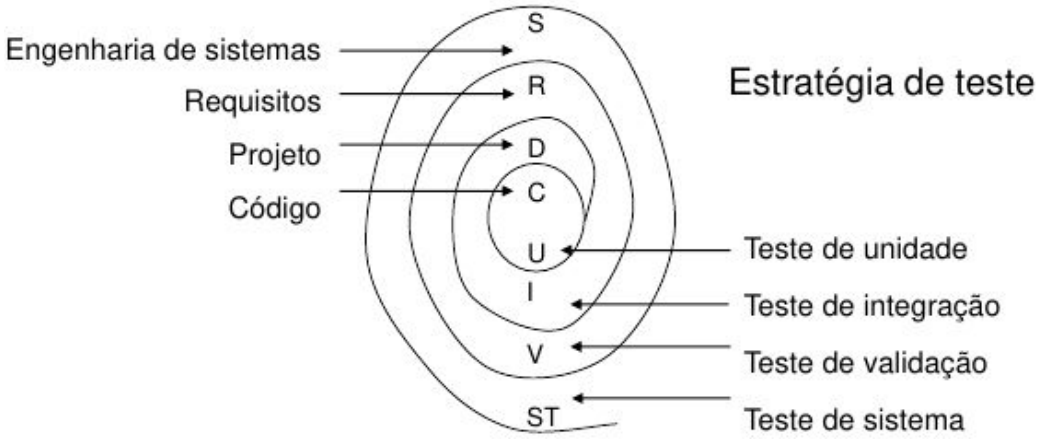
Objetivo: testes de defeitos (busca inconsistências entre o programa e a sua especificação, verificando a correção; é normalmente realizados com protótipos funcionais) ou testes de validação (demonstra ao desenvolvedor e ao cliente do sistema que o *software* atende aos seus requisitos; um teste bem sucedido visa mostrar que o sistema opera conforme especificado pelo cliente.)

Método: testes controlados (realizados em laboratórios ou em condições operacionais sob a supervisão de um testador; são baseados na geração de casos de testes, e podem ser realizados com protótipos funcionais) ou testes estatísticos (avalia robustez, desempenho, confiabilidade etc.; utiliza programas de monitoramento e *logs* com ocorrências operacionais; exemplos de medição incluem número de falhas observadas, tempo de resposta e de execução etc.)

Escopo: testes de unidade de componentes (são conhecidos como testes em ponto pequeno, nele são testadas as unidades individuais, como funções, objetos e componentes) ou testes de integração do sistema (são conhecidos como testes em ponto grande, os componentes são integrados e o conjunto maior é testado, como módulos e sub-sistemas.)

Figura 22.3 Plano de teste como ligação entre o desenvolvimento e os testes.





Abordagens para verificação da correção:

Inspeções analíticas: uma forma estática.

Rastreamento do código (Walkthrough): percorre-se o código executando-o mentalmente, uma forma dinâmica.

Depuração: execução passo-a-passo do código, com visualização de variáveis do programa. É a parte mais imprevisível do processo de teste; erros que indiquem discrepâncias

de 0.01% entre resultados esperados pode demorar uma hora, um dia ou um mês para ser diagnosticado e corrigido.

Testes de correção de programas: testes de unidade, testes de integração.

Prova formal de programas: utiliza métodos formais de desenvolvimento de *software*, como técnicas de especificação, transformação e prova-forma.

Verificação no CMMI:

Preparar Verificação: selecionar produtos de trabalho para a verificação e os métodos de verificação a serem utilizados para cada; estabelecer o ambiente de verificação; estabelecer procedimento e critérios de verificação.

Realizar Revisões Técnicas (peer reviews): preparar as revisões técnicas; realizar revisões técnicas e identificar ocorrências resultantes da revisão técnica; analisar dados relativos à preparação, realização e resultados das revisões técnicas.

Verificar os Produtos de Trabalho Selecionados: executar verificação dos produtos de trabalho face aos seus requisitos especificados; analisar resultados das verificações e identificar ações corretivas.

Validação no CMMI:

Preparar Validação: selecionar os produtos e componentes do produto a serem validados e quais o métodos a serem utilizados para validação de cada um desses; estabelecer o ambiente de validação; estabelecer procedimentos e critérios de validação.

Validar Produto ou Componentes do Produto: realizar a validação dos produtos ou componentes do produto selecionados, de modo a assegurar que estes são apropriados para utilização no ambiente de operação pretendido; analisar resultados da validação e identificar ocorrências.

TIPOS DE TESTE

Teste: conjunto de atividades que pode ser planejado antecipadamente, e realizado sistematicamente. É possível definir *templates*, um conjunto de passos ao qual é possível alocar técnicas de projeto de casos de teste e estratégias de teste específicos.

O processo de teste deve ser revisto continuamente, para permitir uma maior agilidade e controle operacional dos projetos de testes. Testes geram informação sobre qualidade do produto.

Norma IEEE 829-1998: descreve um conjunto de documentos para as atividades de teste. Os documentos cobrem as tarefas de planejamento, especificação e relato de testes.

Equipe de Teste:

Gerenciamento dos Serviços de Testes: gerente de testes; coordenador de testes; auditor de testes.

Execução dos Serviços de Testes: líder de testes; analista de testes; executor de testes.

Inovação dos serviços de testes: engenheiro de testes; arquiteto de testes; automatizador de testes.

Testes Caixa-Preta: conhecendo a função específica que um produto deve executar, testes caixa-preta são realizados para demonstrar que cada função é totalmente operacional. São testes para interfaces; a entrada é adequadamente aceita e a saída é corretamente produzida com a integridade das informações externas mantida.

Testes Caixa-Branca: conhecendo o funcionamento interno de um produto, testes caixa-branca são realizados para garantir que todas as engrenagens, ou seja, toda operação interna de um produto, tem desempenho de acordo com as especificações, e que os componentes internos foram adequadamente postos à prova.

Baseiam-se num minucioso exame dos detalhes procedimentais, por meio da definição de todos os caminhos lógicos possíveis. Isso leva a muitos problemas logísticos, pois o número destes possíveis caminhos pode ser muito grande, levando tempo próximo ao infinito. Ainda assim, esse tipo de teste não pode ser desprezado, podendo-se optar por um número limitado de opções.

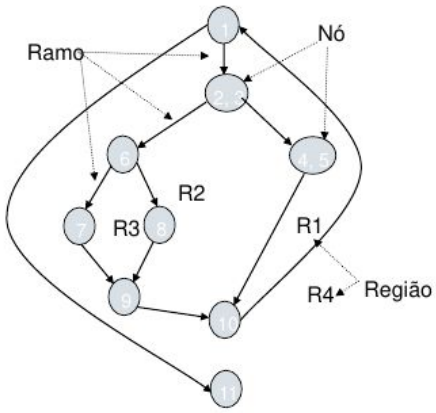
Teste de Caminho Básico: é uma técnica de teste de caixa-branca, que possibilita que o projetista do caso de teste derive uma medida de complexidade lógica de um projeto e use essa medida como guia para definir um conjunto básico de caminhos de execução.

A notação mais simples para representar esse fluxo é o grafo de fluxo de controle, que descreve o fluxo lógico (procedimentos sequenciais, procedimentos partidos por ifs, whiles, cases etc.)

Complexidade Ciclomática: é uma métrica que proporciona uma medida quantitativa da complexidade lógica de um programa. O valor computado da complexidade ciclomática define o número de caminhos independentes do conjunto básico de um programa, e oferece um limite máximo para o número de testes que deve ser realizado para garantir que todas as instruções sejam executadas pelo menos uma vez.

Por exemplo, um conjunto de caminhos independentes, referentes à figura ao lado:

- caminho 1: 1-11
- caminho 2: 1-2-3-4-5-10-1-11
- caminho 3: 1-2-3-6-8-9-10-1-11
- caminho 4: 1-2-3-6-7-9-10-1-11



Grafo de fluxo

Testes de unidade: concentra-se no esforço de verificação da menor unidade do projeto de *software*, o módulo. Baseia-se, quase sempre, na técnica de caixa-branca e pode ser realizado em paralelo para múltiplos módulos.

Testes de integração: o objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto, realizando-se, ao mesmo, testes para descobrir erros associados às interfaces (entradas e saídas entre módulos devem se compatibilizar.)

Testes de validação: são definidas expectativas razoáveis na Especificação de Requisitos de *software*, que descreve todos os atributos do *software* visíveis ao usuário. A validação é bem-sucedida quando o *software* funciona de maneira razoavelmente esperada pelo cliente.

Testes de sistema: uma série de diferentes testes cujo propósito primordial é pôr completamente à prova o sistema.

Teste de recuperação: é um teste de sistema que força o *software* a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada.

Teste de segurança: tenta verificar se todos os mecanismos de proteção embutidos em um sistema o protegerão, de fato, de acessos indevidos.

Teste de estresse: executa o sistema de uma forma que exige recursos em quantidade. Essencialmente, o analista tenta destruir o programa.

Teste de desempenho: idealizado para testar o desempenho de *runtime* do *software* dentro do contexto de um sistema integrado.

Test-Driven Development (TDD): desenvolvimento guiado por testes, prega que só se escreva código novo se um teste falhar; refatorar até que o teste funcione; e nunca passar mais de 10 minutos sem que a barra do JUnit fique verde.

Fake It 'Til You Make It: faça um teste rodar simplesmente fazendo o método retornar constante.

Implementação óbvia: se operações são simples, implemente-as e faça que os testes rodem.

Especificação dos Testes: identificação dos casos de testes que deverão ser construídos ou modificados em função de mudanças solicitadas pelo cliente.

Teste de Recuperação: simular uma operação enquanto há defeito em outras partes do sistema (defeito em impressoras, na rede, em dispositivos de E/S etc.)

Teste de Contingência: disparar processos de instalação emergenciais.

Teste de Configuração: simular uma operação em diferentes sistemas operacionais, em diferentes *hardwares*, em diferentes ambientes.

Teste de Carga e Concorrência: simular duas operações concomitantemente, simular a mesma operação em uma quantidade absurda de vezes simultaneamente.

Teste de Performance: garantir que o tempo de uma operação não ultrapasse certo limite em determinadas condições pré-estabelecidas, e avaliar se isso é respeitado.

Teste de Usabilidade: avaliar se o usuário consegue, pelo programa, solicitar ajuda para realizar certas operações; se as mensagens são claras e objetivas; se o padrão visual é mantido em todos os momentos; se todas as operações possuem caminhos de fuga etc.

Teste de Segurança: impedir operações de acontecerem quando elas não puderem ser realizadas (p. ex., saques com cartões vencidos, acesso a usuários não-identificados, acesso a áreas restritas por usuários sem permissões para aquela área etc.)

Teste Funcional: avaliar se as operações estão impondo os limites supostos (p. ex., se é possível sacar acima do permitido, usando letras em campos somente de números etc.)

Modelagem dos Testes: identificação de todos os elementos necessários para a implementação de cada caso de teste especificado. Pode haver modelagem das massas de testes e a definição dos critérios de tratamento de arquivos (descaracterização e comparação de resultados.)

Preparação do Ambiente: conjunto de atividades que visa a disponibilização física de um ambiente de testes no qual se realizarão as atividades de teste planejadas nas etapas anteriores, de forma contínua e automatizada (sem intervenção humana.)

Execução dos Testes: execução e conferência dos testes planejados, de forma a garantir que o comportamento do aplicativo permaneça em conformidade com os requisitos contratados pelo cliente.

Análise dos Resultados: análise e confirmação dos resultados relatados durante a fase de execução dos testes. Resultados em "não-conformidade" deverão ser confirmados e detalhados para que se realize as correções necessárias. Aqueles em conformidade deverão ter seu resultado positivo reconfirmado.

MÉTODOS ÁGEIS

Surgidos em 2001 após a publicação do *Manifesto For Agile Software Development*. O manifesto ia contra as metodologias de desenvolvimento prescritivas (como o RUP.) Ou seja, se as Metodologias de Desenvolvimento Prescritivas eram rígidas e com alto nível de controle e forte documentação, as metodologias ágeis caminham ao contrário.

Prega: os indivíduos e a interação entre eles como mais importante que processos e ferramentas; o *software* em funcionamento mais que a documentação abrangente; a colaboração com o cliente mais que a negociação de contratos; responder a mudanças mais que seguir a um plano.

Seus princípios básicos são: simplicidade acima de tudo; adaptação rápida e incremental às mudanças; desenvolvimento do software prezando pela excelência técnica; motivação de indivíduos e relação de confiança entre eles como causa de sucesso de projeto; cooperação entre os desenvolvedores e os usuários ou clientes; atender o usuário/cliente, entregando rápida e continuamente produtos funcionais, em curtos espaços de tempo (normalmente duas semanas); *software* funcionando como principal medida de progresso; mudanças no escopo ou nos requisitos do projeto não é motivo de chateação; a equipe de desenvolvimento se auto-organiza, fazendo ajustes constantes em melhorias.

Dá-se alta liberdade à equipe de desenvolvimento, sendo ela quem decide quanto trabalho acredita que pode realizar dentro da iteração, e comprometendo-se a realizar este trabalho.

A qualidade é uma atividade presente nas diferentes metodologias ágeis. A abordagem ágil modificou a forma de desenvolvimento de *software*. Metodologias ágeis também mudaram a forma de atividades de SQA. Documentações não são muito pesadas, mas apenas o que o cliente/usuário necessita. Muitas características já incorporadas na filosofia ágil, como a refatoração, o TDD e a Programação em Par, têm potencial de garantir a qualidade do *software*.

Feature-Driven Development (FDD): metodologia de desenvolvimento de *software* que, seguindo os princípios do Manifesto Ágil, fornece processos para distribuição repetível de *software* como valor para o cliente. É um meio-termo entre o RUP, uma metodologia de desenvolvimento prescritiva, e o XP ou o Scrum, metodologias ágeis.

Características marcantes da FDD incluem: importância dada à qualidade das funcionalidades entregues ao cliente; priorização da entrega de resultados frequentes, tangíveis e funcionais graças ao trabalho dividido em iterações; relatórios de estado e de progresso das atividades; adaptabilidade para projetos e equipes maiores ou menores; desenvolvimento partindo de modelo abrangente.

Extreme Programming (XP): baseada em cinco valores, alguns princípios e várias práticas, que ocorrem no contexto de quatro atividades, destina-se a times de até dez programadores, e a projetos de curto a médio prazo.

Possui quatro fases: codificação (metade do tempo do projeto), testes (um quarto do tempo do projeto), planejamento e projeto (cada uma destas duas últimas sendo um oitavo do tempo do projeto.)

Seus valores são:

Comunicação: é necessária interação entre os membros da equipe, programadores, cliente e treinador para que o projeto tenha sucesso. Assim, para desenvolver o produto, o time precisa de ter qualidade nos canais de comunicação. Preferência a conversas pessoais.

Feedback: respostas às decisões tomadas devem ser rápidas e visíveis. Todos devem ter, o tempo todo, consciência do que está acontecendo.

Coragem: e responsabilidade para alterar código em produção com agilidade e sem causar *bugs*.

Simplicidade: para atender rapidamente às necessidades do cliente, pois quase sempre o que ele quer é mais simples do que aquilo que os programadores constroem.

Respeito: todos têm sua importância dentro da equipe e devem ser respeitados e valorizados; esse respeito mantém o trabalho energizado.

Seus quatro papéis são:

Programadores: foco central da metodologia, sem hierarquia.

Treinador: pessoa mais experiente no time, responsável por lembrar os outros das práticas e valores do XP. Não necessariamente é o melhor programador, mas quem melhor entende a metodologia XP.

Acompanhador: responsável por trazer dados, gráficos e informações que mostrem o andamento do projeto e ajudem a equipe a tomar decisões de implementação, arquitetura e *design*. Algumas vezes, o próprio treinador assume esse papel; em outras, o time escolhe.

Cliente: deve estar sempre presente e pronto para responder às dúvidas dos programadores.

Suas práticas são:

Testes: deve-se preferencialmente escrevê-los antes do desenvolvimento (TDD), e sempre rodando de forma automatizada.

Refatoração: conjunto de técnicas para modificar o código do sistema sem alterar nenhuma funcionalidade. O objetivo é simplificar, melhorar o *design*, limpar, deixar o código mais fácil de entender e de dar manutenção.

Programação pareada: dois programadores sentam juntos no mesmo computador e programam juntos. Enquanto um digita, o outro observa e pensa em melhorias alternativas.

Propriedade Coletiva: o código não pertence a um único programador. Todos da equipe são responsáveis, e todos alteram códigos de todos (mas sempre rodando testes para se certificar de que nada foi quebrado.)

Integração Contínua: depois de testada, cada nova funcionalidade deve ser imediatamente sincronizada entre todos os desenvolvedores. Quanto mais frequente a integração, menores as chances de conflitos de arquivos que vários programadores alteram.

Semana de 40 horas: programar não rende sem que programadores estejam descansados e dispostos. Apenas até 40 horas de trabalho por semana é essencial para a saúde do time.

Cliente sempre presente: cliente é um membro da equipe, e não alguém de fora. Ele deve estar sempre disponível e pronto para atender às dúvidas de desenvolvedores.

Padronizações: se todo time seguir padrões pré-acordados de codificação, mais fácil será manter e entender o que já está feito. O uso de padrões é uma das formas de reforçar o valor "Comunicação", mencionado acima.

Fatores de qualidade aplicados ao SCRUM:

Compatibilidade: cada incremento deve ser testado com tudo que já foi concluído. Iterações rápidas e o desenvolvimento em pequenos times favorecem a criação de componentes compatíveis.

Corretude: uso de boas práticas desde o começo. O cliente sempre está presente, verificando se o andamento está de acordo com os requisitos declarados no *backlog*.

Efetividade de Custo: gráficos *burndown* auxiliam no acompanhamento dos gastos ao longo do tempo do projeto.

Eficiência: o compartilhamento contínuo de informações entre membros da equipe favorecem o uso de artefatos que irão otimizar os recursos de *hardware* e *software*.

Extensibilidade: na reunião de *release*, os objetivos são traçados e, caso tenha surgido algum novo item no *backlog*, as prioridades são alteradas e o escopo do *sprint* é redefinido.

Facilidade de uso: apesar da interação constante com o cliente, a metodologia Scrum não se mostra adequada para lidar com requisitos de usabilidade.

Integridade: a cada reunião de planejamento, cada integrante do time compartilha as lições aprendidas, de sucesso ou não. Seguindo o empirismo do Scrum, o grupo adota novas práticas ao longo do processo.

Manutenibilidade: à medida que há uma interação maior da equipe a cada *sprint*, os erros são detectados mais rapidamente, assim como é possível projetar *software* com maior atenção à integração modular.

Oportunidade: ao final de cada *sprint*, a entrega de uma funcionalidade do produto é realizada. À medida que ocorrem *sprints*, a lista do *backlog* é reduzida e pode ser modificada a prioridade da lista de acordo com as necessidades do cliente.

Portabilidade: o Scrum adota padrões em todas as fases de desenvolvimento do produto, o que permite a criação de arquiteturas independentes de *hardware*.

Reusabilidade: a integração de componentes é facilitada a partir da larga comunicação. Práticas desenvolvidas durante um *sprint* podem ser utilizadas nas próximas iterações a partir das reuniões diárias, de revisão e de retrospectiva.

Robustez: enquanto o produto existir, o *backlog* também existirá. O dono do produto acompanha a execução de todas as iterações e sempre oferece *feedback* sobre o atendimento dos requisitos. Situações não previstas são tratadas rapidamente.

Verificação e validação: reuniões diárias permitem conferir o andamento do trabalho realizado por cada integrante. Como cada iteração significa o entregável ao cliente, o produto é verificado, validado e testado constantemente.

Fatores de qualidade aplicados aos Métodos Ágeis:

Facilidade de uso: casos de uso são desenvolvidos colaborativamente, com *feedback* do cliente ao longo do projeto, proporcionando ajustes para adequar a usabilidade.

Integridade: o gerente de projeto trabalha em conjunto com toda a equipe a fim de identificar a alta prioridade de resolver os itens de trabalho da lista de itens. O plano de iteração anterior inclui uma avaliação dos resultados, e também pode ser usado como entrada para o planejamento da iteração atual. No âmbito deste processo, o gerenciamento de configurações mantém consistentes as versões de artefatos e suas configurações.

Manutenibilidade: gerenciamento de mudança fornece dados para medir o progresso e proporciona um meio eficaz para se adaptar às mudanças e problemas. As versões de todo trabalho são mantidas atualizadas, e as alterações são geridas por tarefas de solicitação de mudança que são posteriormente priorizadas em uma lista de itens de trabalho.

Oportunidade: atende este fator por realizar desenvolvimento iterativo e incremental, permitindo entrega rápida e realizando curtos ciclos.

Portabilidade: o uso de padrões de desenvolvimento praticado no mercado separa o *software* em camadas, permitindo a independência e autonomia de *hardware* e *software* nesta metodologia.

Reusabilidade: como o apoio de documentação adequada, é possível fazer o reuso de código e adaptar, de forma simples, a diferentes processos e aplicações, reduzindo o tempo de desenvolvimento, e garantindo este fator de qualidade (a reusabilidade.)

Robustez: ao longo das iterações, a disciplina Arquitetura se relaciona com outras disciplinas, como: requisitos ao obter arquitetura significativa, testes ao verificar a estabilidade e corretude, entre outras, para alcançar uma arquitetura robusta para o sistema.

Verificação e validação: testadores são responsáveis por realizar testes de funcionalidade, usabilidade, confiança, performance e suportabilidade várias vezes por iteração, a cada vez que a solução for incrementada com o desenvolvimento de algo novo, mudança ou correção de erro, a fim de retirar os ricos mais cedo no ciclo do sistema e garantir a qualidade de software.