Um processo são um conjunto de atividades, normalmente agrupadas em fases, com lógica e estruturadas, de forma a serem de execução sistemática, sequencial e automatizada durante o desenvolvimento de software.

Um método são os conjuntos de técnicas que se usam para se desenvolverem os diferentes processos, serve para modelar a atividade que pretendemos desenvolver.

Os métodos estão subentendidos nos processos. Para se adoptar uma determinada metodologia é necessário ter em atenção o nosso processo, os nossos métodos e devemos de dominar a nossa ferramenta de trabalho; também é necessário conhecermos o ambiente e o contexto para/onde o nosso software está a ser desenvolvido.

Estes 4 conceitos base, são aqueles que se deve de ter em atenção quando se desenvolve software. É de extrema importância conhecer o domínio do problema. Quanto melhor se conhecer o domínio do problema, mais rápido se consegue modelar o mesmo, identificar os seus processos constituintes e assim, usando a engenharia conseguindo, sistematizar, uniformizar e automatizar o processo e dar uma solução a este. O grande objetivo de uniformizar, sistematizar e automatizar o problema (aplicar os conceitos fundamentais de engenharia) é garantir a qualidade de software. É possível abordar esta temática através de uma perspectiva do produto ou do processo (normas ISO/IEC 9126 , TickIT + ISO 900 + CMMI respectivamente) (Ambas as perspectivas usam o SQA, Software Quality Assurance, que consiste em verificar o software, rever o software e a sua validação). Para começar a modelar um software, que apresente uma solução para o problema e garantir a sua qualidade, normalmente, desenvolve-se um S.R.S (System Requirements Specifications). Após, aplica-se modelos (métodos) para se conseguir desenvolver uma solução, uniformizada e sistematizada. Deve-se seguir as normas, apesar de, para alguns engenheiros, técnicos e designers, ser um “inibidor da criatividade“, assim garante-se que os software irá ter qualidade. Só após, existir um modelo de uma solução, é feita a escolha, de uma/várias ferramenta(s), mais indicada(s), para construir o nosso programa. Sendo assim um software que tenha qualidade elevada é pós-ativo no processo, preventivo nos problemas e defeitos e cumpre todos os requisitos (funcionais e não funcionais) do S.R.S. . Podemos concluir que, para garantir a qualidade de software, deve-se garantir a qualidade de produto e a qualidade de processo. Não há garantia quanto à definição da qualidade de software. Existem várias, de vários autores e entidades: \* “Ausência de falhas no produto“ (Juran, 1988); \* “Satisfaz o cliente e vai de encontro com as suas necessidades“ (Juran, 1988); \* “Estar em conformidade com os requisitos, prevenção“ (Corsby, 1986); \* “(…) totalidade das propriedades e características de um produto (…) que o tornam apto a satisfazer as necessidades implícitas e explicitas“ (ISO-8402); \* “(…) satisfação total do cliente“ (I.B.M.); \* “Todas as características cumprem os requisitos“ (ISO 9001-00). Segundo o software engineering body of knowledge ou SWeBoK a qualidade de software está completamente dependente dos requisitos, estes influenciam as suas características de qualidade e os métodos de cálculo e critérios de aceitação para avaliar estas características. Segundo o SWeBoK a qualidade de software esta dividida em 3 campos: \* Fundamentos: \* A ética e a cultura da engenharia de software (desenvolvido pelo IEEE); \* Os gastos: (Uma maior prevenção significa, menores defeitos de qualidade, menores custos de prevenção e menores custos de produção) \* Gastos de prevenção; \* Gastos de avaliação; \* Gastos de falhas internas e externas. \* Modelo e qualidade das características; \* Melhorias. \* Gestão de qualidade: (Se usar a gestão de qualidade consegue-se diminuir os custos através de prevenção, que têm um custo associado mas mais baixo) \* Garantir (S.Q.A. : Verificar, Rever e Validar); \* Rever; \* Auditorias. \* Considerações: \* Gestão de qualidade; \* Medidas de qualidade; \* Aplicar requisitos de qualidade; \* Caracterizar os defeitos. As medidas para garantirem a qualidade incluem a estruturação de um processo de desenvolvimento com métodos, técnicas e ferramentas. Os programas de gestão de qualidade podem incluir: \* Documentação de padrões de códigos; \* Métodos; \* Ferramentas; \* Procedimentos de recuperação de dados; \* Gestão de configurações; \* Documentação dos documentos encontrados; \* Rastreabilidade (GIT). Infelizmente, alta qualidade de software traduz-se em alto custo. O SWeBoK, defende que a qualidade do processo e a do produto estão interligados (como referido a cima), a qualidade do processo influenciam as características do produto de software que por sua vez afectam a “quality-in-use“ presenciada pelo cliente. A qualidade pode ser melhorada através de um processo de revisão, avaliação e de melhorias iterativo e continuo que necessita de um sistema de controlo de versões (GIT) (ciclo de vida do software) e prevenção de erros mais a sua remoção Os standards que regem a qualidade do processo são: \* TickIT; \* ISO 9001-00; \* CMMI. O standard da qualidade de produto é a ISO 9126-01. A qualidade total ou a T.Q.M. (total quality management), depende: \* da Mão de obra (o quanto qualificada é), \* das Melhorias constantes (nos processos); \* e no Foco no cliente. que resulta em: \* Métricas; \* Modelos; \* Medições; \* Analises. O total quality management possui vários níveis: 1. Inspeção: \* Consiste em verificar o S.R.S. . 2. Controlo de qualidade: \* São o conjunto de inspeções, revisões e testes utilizados durante o ciclo de desenvolvimento para assegurar que cada produto cumpre os requisitos previstos (medições e feedbacks). 3. Garantia de qualidade; 4. Qualidade Total

A área de analise de requisitos de software ou software requirements knowledge area ou ainda K.A. preocupa-se com procura de requisitos, analise, especificação e validação de requisitos; assim como a sua gestão durante o “tempo de vida“ de desenvolvimento de software. De acordo com estudiosos, analistas e industriais, normalmente o desenvolvimento de software é vulnerável quando os requisitos não são bem especificados. O termo “engenharia de requisitos“ refere-se a aplicar os conceitos básicos de engenharia ao desenvolvimento de requisitos. Normalmente é comum acontecer um “efeito bola de neve“ quando não se especificam requisitos, todo o processo de desenvolvimento de software vai ser defeituoso. Decompor o nosso processo em requisitos ajuda bastante a arquitetar uma boa solução. Definição de um requisito de software: “At its most basic, a software requirement is a property that must be exhibited by something in order to solve some problem in the real world“ - na sua forma mais básica um requisito de software é uma propriedade que o software possui que dá uma solução a um problema; normalmente são complexos, pois procuram dar resposta a um problema do “dia á dia“. Uma das característica de um requisito é que ele se traduz numa “feature“ ou num requisito funcional ou ainda não funcional que consegue ser testado individualmente. Às vezes, validar requisitos pode ser penoso, por exemplo um software de um “call center“ implica um “software de simulação“ para testar os seus requisitos. Os engenheiros de software devem de ter esse facto presente também quando desenvolvem software. Os requisitos também podem ter comportamentos, e outras características associadas. Definição de requisito de produto e processo: “A product requirement is a need or constraint on the software to be developed.“ - um requisito de produto é a necessidade da criação de condições num software. “A process requirement is essentially a constraint on the development of the software“- um requisito de processo é um condição num software. Podemos então resumir que um requisito do produto é a necessidade da criação de uma condição e um requisito de processo a criação efetiva dessa condição. Alguns requisitos de software criam requisitos implícitos e explícitos de desenvolvimento de software. Definição de requisitos funcionais e requisitos não funcionais de software: “functional requirements describe the functions that the software is to execute“ - os requisitos funcionais são as funções que um software tem de executar, são as features de um software. Também podem ser descritos como a parte do software, a qual, um conjunto finito de testes, podem ser desenvolvidos. “Nonfunctional requirements are the ones that act to constrain the solution.“ - os requisitos não funcionais de um software são as restrições para o desenvolvimento de uma solução, também podem ser chamados de requisitos de qualidade de um software. Este tipo de requisitos pode ser divido em muitos sub-temas. Alguns softwares também possuem propriedades emergentes, o que significa que são requisitos que dependem uns dos outros, ou seja só se consegue analisar se de olhar para o software como um todo. Estas propriedades emergentes são extremamente dependentes da arquitetura de software. Sistema na engenharia de requisitos significa software, hardware, firmware, utilizadores, informação, técnicas, e outros elementos de suporte. Como definido pelo I.N.C.O.S.E. um requisito de sistema são os requisitos do sistema como um todo, num sistema contendo componentes de software, os requisitos de software, são uma derivação dos requisitos de sistema. A escolha dos requisitos têm de ter em atenção temas como os objetivos, conhecimento do domínio, uma diretivas das partes interessadas no desenvolvimento do software, as regras do negócio, o meio de operação do software e o meio organizacional para onde o software está a ser desenvolvido. As técnicas para se fazer uma escolha adequada incluem as entrevistas, descrição de cenários, desenvolvimentos de protótipos (normalmente estão ligados á descrição de cenários), observação do meio organizacional, histórias dos utilizadores ou críticas (por exemplo: “As a <role>, I want <goal> so that <benefit>) e ainda podem incluir a analise de produtos semelhantes entre outras…

Os testes software, são uma forma dinâmica de verificar se o programa, através de um conjunto de testes finito, age de acordo com o esperado. Executar testes de software de forma dinâmica, significa executar testes para uns determinados valores de entrada, pré-definidos. Existem, também, formas de testes estáticas, que são usadas em testes de qualidade de software. É importante considerar que, às vezes, os valores de entrada, (sozinhos), podem ser insuficientes para concluir se o software opera de acordo como o planeado. Mesmo um pequeno programa possui um número finito de testes possíveis de serem executados. Realizar testes pode ser uma tarefa exaustiva, daí na prática o número de testes a executar é infinito, por esse motivo, são normalmente usados critérios de restrição aos números de testes. Esses critérios são, às vezes, difíceis de escolher. Quanto melhor forem os critérios mais eficazes serão os testes, mas, como foi dito anteriormente, essa escolha é difícil de se fazer; usam-se, estratégias de analise de risco e recorre-se a técnicos especializados de desenvolvimento de software para auxiliar as equipas de teste a restringir as condições de teste. Quando se realizam testes, esperam-se, uns determinados valores de saída, conforme os valores de entrada. Atualmente, está a haver uma mudança no pensamento das equipas de desenvolvimento de software em relação aos testes. No passado, os testes de software só eram efetuados no final do desenvolvimento do software, em contraste, hoje em dia, os testes de software são uma parte crucial do ciclo de vida do desenvolvimento do programa. O S.W.E.B.O.K. defende que o planeamento dos testes deve de começar logo nas fazes primordiais de desenvolvimento de software, na escolha dos requisitos do processo, assim garantido também a qualidade deste. É importante fazer referência, à propriedade iterativa dos testes, não são estanques, mas sim um processo evolutivo e que acompanha o desenvolvimento do produto, não deve de ser estático. Para muitas empresas a melhor abordagem é garantir a prevenção de erros. Então podemos concluir que os testes de software são uma abordagem preventiva e focada na qualidade final, são um bom indicador e uma boa forma de desenvolvimento de software. Claro que mesmo assim podem haver erros no produto final, é algo que as equipas de manutenção de software devem de abordar. Existem técnicas estáticas e técnicas dinâmicas para os testes de software.

É importante lembrar que, nem todos os métodos descritos são aconselháveis para todos os softwares e nem todos as técnicas de teste foram listadas. Para identificar que tipo de teste efetuar, o “test target“, ou seja, os objetivos e os critérios, são o suficiente. Para saber quantos testes são necessários realizar para alcançar o nosso objectivo, usa-se o “critério de adequação software“, através deste conseguimos saber quantos testes temos de realizar para conseguir cumprir o nosso objetivo. Para se conseguir responder à pergunta, que testes efectuar, usam-se “critérios de seleção“, assim consegue-se ter mais ou menos uma noção de o quê testar e como. 1. ERROS vs DEFEITOS: Termos com o intuito de distinguir o problema no nosso software, e a sua origem (o defeito e o erro, respetivamente). Na teoria, há erros que podem não afetar o nosso software; os “problemas“ no desenvolvimento de software funcionam de forma reactiva e não pró-ativa, se ninguém encorar o defeito, o software não tem defeitos. Por esse mesmo motivo é que é importante, sempre que se encontram defeitos, remover os mesmos. Também é possível uns erros estarem a encobrir outros e uns defeitos a encobrir outros também. É necessário ter em muita atenção as primeiras fases de desenvolvimento de software; se não conhecer bem o domínio, o negócio, as intenções das partes envolventes, etc…podem haver erros de raciocino que levam a defeito. É tudo relativo, não há nenhuma norma ou convenção para especificar que terminologia usar, para simplificar iremos considerar que os defeitos são os famosos “bugs“ e os erros, são erros no raciocínio que provocam defeitos. O S.W.B.O.K. diz que existem erros, faltas, fracassos, defeitos…e mais. 2. Critérios de seleção de testes e critérios de adequação de software: Como dito a cima, critérios de seleção de testes servem para selecionar que tipos de metodologia de teste deve de se usar relativamente ao que pretendemos testar e ao software que temos. Critérios de adequação de software, são critérios, que os engenheiros de software e as equipas de teste escolhem para poder efectuar os testes. 3. Eficácia e Objetividade: A eficácia do teste é determinada pela observação de alguns testes e a sua performance. Os testes a efetuar dependem dos objetivos, dito isto a eficácia dos testes está dependente dos objetivos, quanto mais especificados os objetivos mais eficaz será o nosso teste. 4. “Testing for defect discovery“ ou Testar orientado a erros: Um teste bem sucedido é aquele que faz com que o teste falhe (nota o objetivo dos testes é falhar), ou seja o objetivo deste teste é, ambiente realístico, conseguir testar o sistema, de forma a que não haja nenhum defeito, e demonstrar que o software cumpre os requisitos e os objectivos traçados. 5. “The oracle problem“ ou teste oracle: um “oracle“ é um ser humano, ou algo mecanizado que averigua, de acordo com se o software “passa“ ou “falha“ de acordo com uns requisitos específicos. Existem muitos tipos de teste oracle, pode-se fazer um teste para ver se ele cumpre os requisitos, para ver se o programa vai de acordo com uns comportamentos esperados numa determinada situação, ou ainda, se a documentação é a correta. Fazer um “oracle“ automatizado é dispendioso, não é uma opção viável para equipas com um orçamento pequeno. 6. Limites práticos e teóricos dos testes de software: A teoria dos testes alerta para o facto de não ser bom, numa serie grande de testes, serem todos bem sucedidos. “Unfortunately, most established results of testing theory are negative ones, in that they state what testing can never achieve as opposed to what is actually achieved.“ Isto significa que, por exemplo o algoritmo Dijkstra para o aforismo (regras) do software, é capaz de encontrar defeitos, mas não funciona se não houver defeitos, então é necessário efetuar testes com cuidado e também usar técnicas de gestão de risco, o S.W.E.B.O.K. diz que os testes se fazem através de risco. 7. O problema dos caminhos impossíveis: Caminhos impossíveis são caminhos que não conseguem existir, independente dos valores de entrada. São um grande problema no design de testes com base em mapeamento de caminhos, especialmente se esses mecanismos de testes forem automatizados. 8. Testabilidade: O termo a “testabilidade do software“ em engenharia de software pode ter dois significados: pode significar que o software é fácil de testar tendo em conta os critérios definidos para o testar ou ainda a possibilidade de medir, estatisticamente, a probabilidade que um caso de testes tem de falhar ou ser bem sucedido. Ambos os significados são importantes.

Fatores Explícitos: \* Exatidão das estimativas – considera a extensão do alcance das estimativas do Projeto de Teste (prazo, custo e esforço); \* Eficiência - quantidade de recursos requeridos pelo Projeto de Teste para desempenhar alcançar seu objetivo; \* Manutenibilidade – permite localizar e remover um defeito e implementar modificações específicas em um Caso de Teste do Projeto de Teste; \* Testabilidade – possibilita assegurar que os Casos de Teste do Projeto de Teste são confiáveis e não apresentarão falhas durante a sua execução; \* Flexibilidade – corresponde ao esforço necessário para modificar um Caso de Teste do Projeto de Teste; \* Portabilidade – permite a transferência dos Casos de Teste de um Projeto de Teste de uma plataforma de hardware e ou software para outra; \* Reusabilidade – relaciona-se ao uso de um Caso de Teste de um Projeto de Teste em outros Projetos de Teste, ou seja, ao empacotamento e escopo dos requisitos que o Caso de Teste verifica; e \* Estabilidade – é a extensão em que os fatores acima são mantidos ao longo da vida útil de cada Caso de Teste do Projeto de Teste. Fatores Implícitos: \* Prazo – é o prazo de entrega do produto do Projeto de Teste esperado pelo cliente, o qual pode ser definido em termos contratuais; \* Informações sobre progresso – permite que o cliente receba informações a respeito do andamento do Projeto de Teste. A periodicidade pode ser definida em contrato; \* Cobertura funcional – corresponde à capacidade dos Casos de Teste do Projeto de Teste em relação ao teste dos requisitos de negócio exigidos da Aplicação de TI pelo cliente; \* Confiabilidade – corresponde à capacidade de cada um dos Casos de Teste do Projeto de Teste em manter seu nível de desempenho, produzindo a evidência esperada; \* Integridade - relaciona-se com o nível de controle de acesso ao Caso de Teste por pessoas não autorizadas; \* Usabilidade – é o fator requerido para aprender, operar, preparar entradas e interpretar as saídas de cada um dos Casos de Teste do Projeto de Teste, do ponto de vista do seu executor; \* Retorno sobre o investimento – corresponde aos benefícios econômicos obtidos pelo cliente por meio do Projeto de Teste; e \* Tempo de atendimento – relaciona-se ao tempo de espera do cliente para receber as Evidências do Projeto de Teste.

O objecto dos testes é chamado de test target e o seu objectivo de objective. O target pode variar. Podemos dizer que se pode realizar teste de software a 3 níveis distintos e um não é mais importante que outro, nem implicam modelos diferentes: \* UNITÁRIOS: testes unitários realizam-se a uma parte especifica do código, ou a programas mais pequenos, a uma “unidade“ de código, ou até a grandes sistemas compostos de várias unidades de código. Normalmente, como os testes unitários requerem a analise de código, estes testes são realizados, mas nem sempre, pelos programadores; \* INTEGRAÇÃO: testes de integração servem para testar a integração de diferentes componentes de software. Algumas estratégias como analise “de baixo para cima“ e “de cima para baixo“ usam-se em softwares de design hierárquico ou sistemático. Atualmente as integrações em componentes é incrementada, crescente, conforme o desenvolvimento do software, usam-se tecnologias de partilha de threads; os engenheiros de software estão, então, constantemente a efectuar testes de integração. Há uma forma de efectuar testes, que se chama testes “big-bang“ aonde se juntam todos os componentes do software de uma vez; \* SISTEMAS: como o nome indica estes testes de software efetuam-se ao sistema num todo. Aonde os testes unitários e os testes de integração ao serem realizados apontam muitos defeitos, estes são realizados para testar se os requisitos não funcionais do sistema estão a ser cumpridos (teoria de desenvolvimento de software de forma dinâmica). Normalmente U.I. (user interface) são avaliados neste teste, hardware entre outros. Os casos de teste podem ser feitos para testar se especificações funcionais do software estão corretamente implementadas, que na literatura são referidas como: \* Testes de conformidade; \* Testes de de funcionais; \* Testes de correção. Também especificações não funcionais podem ser testadas como se o software é “user-friendley” ou a sua performance entre outros atributos. Outro grande objetivo para os testes de software são as medidas de “confiança“ (reliability), a identificação de medidas de segurança, medidas de usabilidade, etc…para os quais medidas diferentes de testes são tomadas. Resumido, os vários tópicos listados a baixo, são os mais comuns (na literatura) para o qual as equipas de testes definem os seus objetivos: \* Testes de aceitação ou de qualificação: determina se um sistema cumpre os seus critérios de aceitação. Depende dos requisitos, realizam-se, testando se os comportamentos que o programa tem vão de acordo com os desejados pelos utilizadores. Este tipo de teste pode ou não envolver programadores; \* Testes de instalação: Estes testes realizam-se para testar como um sistema opera sob um determinado ambiente target. A forma como se instala também pode ser analisada. \* Testes alfa e beta: testes alfa acontece quando se fornece o software a um pequeno grupo de utilizadores previamente escolhidos para apontarem os defeitos do produto. Testes beta acontece quando se fornece o produto a uma amostra maior de utilizadores, também com o objetivo de reportar os erros no produto. Normalmente não são referidos no plano de testes porque não são controláveis. \* Testes de confiança: fazer testes de software aumenta a confiança no mesmo, especialmente se os dados estatísticos forem favoráveis ou se os testes efetuados forem aleatórios de acordo com os objectivos traçados. \* Testes de regressão: conforme o desenvolvimento do software, os testes anteriormente feitos, se efetuados novamente são validos. Se o software for incremental então isto pode acontecer. São uma boa prática; \* Testes de performance: servem para testar se o software cumpre uns determinados requisitos de performance; \* Testes de segurança: os testes de segurança focam-se em saber se o software está protegido contra ataques exteriores: “security testing verifies the confidentiality, integrity, and availability of the systems and its data“. Normalmente testes de segurança envolvem negative testing; \* Testes de stress: tem como objetivo testar o software na sua “máxima“ carga e até para além da sua máxima carga para apurar se os mecanismos de segurança do software existem e estão funcionais; \* Testes de back-to-back: a norma IEEE/ISO/IEC standard 24765 define que testes back-to-back fazem-se testando os mesmos valores de entrada para o mesmo programa corrido em duas máquinas distintas ou duas vezes seguidas e analisar o erros nos valores de saída; \* Testes de recuperação de dados: são testes que testam a capacidade do sistema de recuperação após um “crash“; \* Testes de interface: erros de interface acontecem em sistemas complexos. Normalmente os casos de teste são definidos pela interface; \* Testes de configuração: são testes que testam como o software funciona com várias configurações de vários utilizadores; \* Testes de usabilidade e de interação humana.

As técnicas descritas a seguir tem como objectivo sistematizar ao máximo o processo de testes de software, “partindo“ o processo de testes para este ser o mais simples possível. A sua classificação é efetuada de acordo com o tipo de testes que este efetua. Estas técnicas são normalmente denominadas como: \* Técnicas de “caixa branca“ (ou também de “caixa de vidro“) se os testes efetuados forem sobre a informação de como o software foi codificado ou desenhado (testes estruturais); \* Testes de “caixa preta“ se os testes forem aplicados aos valores de entrada e aos valores de saída exclusivamente (testes funcionais). Encontram-se, listados a seguir, os exemplos de técnicas que usam estas duas abordagem: (baseados na experiência do engenheiro de software e na sua intuição) \* Ad hoc: é uma técnica que só se baseia na experiência do engenheiro de software, é util para identificar casos de teste que normalmente técnicas mais formalizadas não conseguem identificar; \* Teste exploratório: é uma abordagem dinâmica que depende do conhecimento do engenheiro de software, aonde o planeamento dos testes e a sua execução é simultânea. Não requer um planeamento de teste prévio. (testes de caixa preta ou testes funcionais ou testes de foco em valores de entrada e valores de saída) \* Classe de equivalência: esta técnica consiste em “partir“ os dados de entrada em classes de equivalência, de acordo com critérios específicos. Quando o se insere valores inválidos, o programa deve de mostrar uma mensagem de erro. Normalmente, para cada grupo de testes é efetuado um número especifico de classes de equivalência; \* Combinação de pares: este teste realiza-se ao combinar valores “interessantes“ para cada par de valores de entrada em vez de considerar toda a gama de variáveis possíveis. Este tipo de teste faz parte do grupo de testes combinatórios; \* Analise dos valores de limite: os casos de teste são escolhidos perto dos valores limite do domínio das variáveis de entrada, de acordo com a teoria que normalmente as falhas ocorrem nos valores limites, ou nos valores próximos. Uma variante deste teste são os testes de robustez; \* Testes aleatórios: é necessário conhecer o limite do domínio, após de o conhecer, consegue-se automatizar facilmente este tipo de testes, porque resulta em testar valores de entrada aleatórios. Teste Fuzz ou Fuzzy são uma variante dos testes aleatórios mas focada em testes de segurança. (testes de caixa branca ou de caixa de vidro ou testes estruturais) \* Critérios de controlo de fluxo: modelos de controlo de fluxo tem como objetivo analisar todas as condições, blocos de condições ou combinação de condições do programa. O critério mais forte do controlo de fluxo é o teste por caminhos que tem como objetivo testar todas as combinações possíveis de caminhos formados por essas condições; \* Critérios de fluxo de informação: o grafo de controlo de fluxo contem informação de como as variáveis são definidas usadas e depois descartadas;