Artigo Engenharia de Software 3 - Requisitos Não Funcionais

Requisitos Não Funcionais e Arquitetura de Software. A organização das funcionalidades de software de um sistema é explicitada através da arquitetura de software. Artigo da Revista Engenharia de Software edição 3.

Arquitetura de Software

A organização das funcionalidades de software de um sistema é explicitada através da arquitetura de software. Há uma grande quantidade de estilos arquiteturais que trazem características peculiares a cada estilo, e estes, por sua vez, podem ser combinados gerando novos estilos. A heterogeneidade de estilos arquiteturais é salutar. Na verdade, isto ocorre devido à necessidade da arquitetura prover suporte a um conjunto de requisitos, às vezes conflitantes, dentre eles os **requisitos não funcionais**, tema abordado neste artigo.

Conteúdo relacionado: Cursos de Engenharia de Software

Requisitos Não Funcionais e Arquitetura de Software

O projeto da arquitetura de software é uma etapa essencial no desenvolvimento de sistemas de software de grande porte e complexos. Dentro deste contexto, a arquitetura de software é fundamental para o desenvolvimento de linhas de produção de software onde se tem um conjunto de funcionalidades concebidas e implementadas a partir da mesma arquitetura (de software) base. Entretanto, antecedendo a etapa de projeto da arquitetura de software, há a necessidade de fazer o levantamento dos requisitos do sistema.

De um modo geral, o conjunto de requisitos de um sistema é definido durante as fases iniciais do processo de desenvolvimento. Tal conjunto de requisitos é visto como uma especificação do que deveria ser implementado. Os requisitos são descrições de como o sistema deveria se comportar, e contêm informações do domínio da aplicação e restrições sobre a operação do sistema.

Durante a fase de elicitação de requisitos, um projetista ou arquiteto de software faz uso de sua experiência a fim levantar os requisitos, buscando identificar características do sistema a ser desenvolvido. Além disso, informações do domínio juntamente com informações de estilos

arquiteturais existentes podem ser utilizados como fontes de dados que auxiliam na identificação dos requisitos.

Outro recurso que pode ser usado pelo projetista é construir cenários. Os cenários de uso oferecem suporte a requisitos específicos e visam tanto a elicitação quanto a análise de requisitos. Uma vez que o conjunto de requisitos tenha sido obtido, o projetista/arquiteto de software estará em condições de iniciar o projeto da arquitetura de software, conforme ilustrado na **Figura 1**.

Este processo de levantamento e análise de requisitos, em conjunto com o uso de cenários, é usado no suporte da definição da arquitetura de software, como é discutido ao longo do artigo. É importante observar que a etapa de projeto arquitetural pode precisar fazer uso de cenários de uso ou mesmo uma re-análise a fim de refinar a arquitetura a ser empregada no sistema a ser desenvolvido.

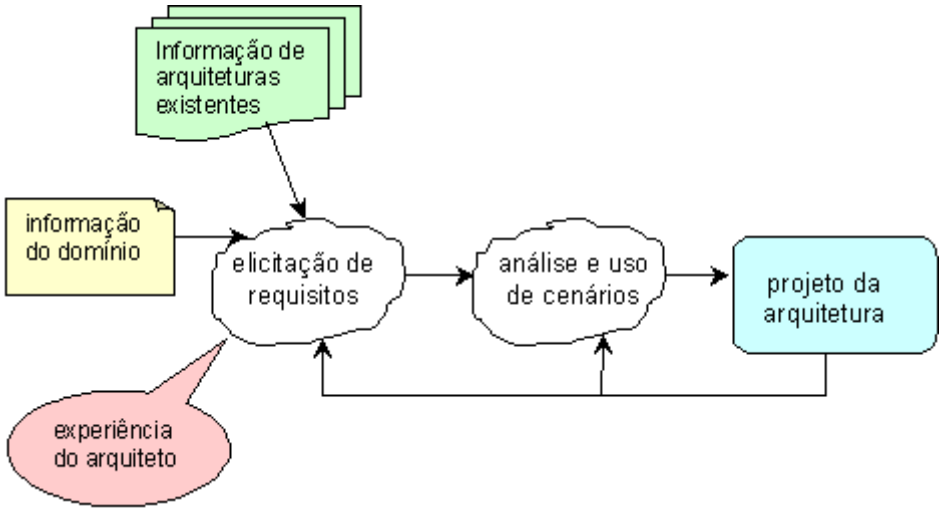

Figura 1. Elicitação de requisitos.

O processo de desenvolvimento baseado na arquitetura considera a arquitetura de software como fator orientador do processo. Isto acarreta em **colocarmos os requisitos não funcionais** associados à arquitetura como principais aspectos do processo de desenvolvimento. Note que o desenvolvimento de um sistema de software centrado na arquitetura inicia-se com um arquiteto de software, de posse de um conjunto de requisitos do sistema. Nesse momento, busca-se identificar qual estilo ou combinação destes oferece suporte mais adequado a esses requisitos e, portanto, derivar uma arquitetura de software que atenda às características do sistema a ser desenvolvido. Vale ressaltar que a complexidade de um sistema de software é determinada tanto por seus requisitos funcionais – o que ele faz – quanto **requisitos não funcionais** – como ele faz. Tal distinção é baseada nas seguintes definições

Confira nosso Guias Engenharia de Software

Requisito funcional

Um requisito de sistema de software que especifica uma função que o sistema ou componente deve ser capaz de realizar. Estes são requisitos de software que definem o comportamento do sistema, ou seja, o processo ou transformação que componentes de software ou hardware

efetuam sobre as entradas para gerar as saídas. Esses requisitos capturam as funcionalidade sob o ponto de vista do usuário.

Requisito não funcional

Em engenharia de sistemas de software, um requisito não funcional de software é aquele que descreve não o que o sistema fará, mas como ele fará. Assim, por exemplo, têm-se requisitos de desempenho, requisitos da interface externa do sistema, restrições de projeto e atributos da qualidade. A **avaliação dos requisitos não funcionais** é feita, em parte, por meio de testes, enquanto que outra parte é avaliada de maneira subjetiva.

Perceba que tanto os requisitos funcionais quanto os **não funcionais** possuem importância no desenvolvimento de um sistema de software. Entretanto, **os requisitos não funcionais**, também denominados de atributos de qualidade, têm um papel relevante durante o desenvolvimento de um sistema, atuando como critérios na seleção e/ou composição de uma arquitetura de software, dentre as várias alternativas de projeto.

Cabe salientar que à medida que os sistemas tornam-se maiores e mais complexos, o **suporte a requisitos não funcionais** depende cada vez mais de decisões tomadas no projeto da arquitetura de software. Trata-se de uma visão compartilhada pelos profissionais da área e, especificamente, pela comunidade de arquitetura de software.

Requisitos Não Funcionais

Requisitos não funcionais são aqueles que não estão diretamente relacionados à funcionalidade de um sistema. O termo requisitos não funcionais é também chamado de atributos de qualidade. Os requisitos não funcionais têm um papel de suma importância durante o desenvolvimento de um sistema, podendo ser usados como critérios de seleção na escolha de alternativas de projeto, estilo arquitetural e forma de implementação. Desconsiderar ou não considerar adequadamente tais requisitos é dispendioso, pois torna difícil a correção uma vez que o sistema tenha sido implementado. Suponha, por exemplo, que uma decisão tenha sido feita de modularizar a arquitetura de um sistema de modo a facilitar sua manutenção e adição de novas funcionalidades. Entretanto, modularizar um sistema adicionando uma camada a mais pode comprometer um outro requisito, o de desempenho. Portanto, faz-se necessário definir logo cedo quais requisitos não funcionais serão priorizados na definição de uma arquitetura.

Os requisitos não funcionais abordam aspectos de qualidade importantes em sistemas de software. Se tais requisitos não são levados em consideração, então o sistema de software poderá ser inconsistente e de baixa qualidade, conforme discutido acima. Para tanto, quanto mais cedo forem definidos os critérios arquiteturais, mais cedo o projetista pode identificar o estilo, ou combinação de estilos, mais apropriado ao sistema considerado.

Ao desenvolver um novo sistema de software bem como sua arquitetura, os projetistas ou engenheiros de software apresentam um conjunto de atributos de qualidade ou requisitos não

funcionais que o sistema deveria suportar. Exemplo destes requisitos são desempenho, portabilidade, manutenibilidade e escalabilidade.

A arquitetura de software deveria oferecer suporte a tais requisitos. **sto resulta da associação existente entre arquitetura de software e requisitos não funcionais**. Importante observar que cada estilo arquitetural (isto é, a forma na qual o código do sistema é organizado) suporta requisitos não funcionais específicos. A estruturação de um sistema é determinante no suporte oferecido a um requisito não funcional. Por exemplo, o uso de camadas permite melhor separar as funcionalidades de um sistema, tornando-o mais modular e facilitando sua manutenção.

Considere, por exemplo, o padrão IEEE-Std 830-1993 [IEEE 1993] que lista um conjunto de 13 requisitos não funcionais a serem considerados no documento de especificação de requisitos de software. Este padrão inclui, dentre outros, requisitos de desempenho, confiabilidade, portabilidade e segurança.

Embora haja um conjunto de propostas, consideradas como complementares, concentraremos nossas atenções num conjunto de requisitos diretamente associados a um sistema de software e, especificamente, à arquitetura de software. Este conjunto é baseado numa classificação apresentada por Sommerville, onde é feita a distinção entre requisitos externos, de produto, e de processo [Sommerville 2007].

A **Figura 2** é uma adaptação da proposta de Sommerville, onde consideramos os requisitos de produto, associados à arquitetura de software, bem como adicionamos outros não presentes na proposta original de Sommerville. É importante observar que a **Figura 2** mostra um **subconjunto de requisitos não funcionais**, denominados de requisitos de produtos os quais estão associados à arquitetura de um sistema de software. Note que a classificação apresentada em [Sommerville 2007] ainda considera os requisitos de processo e requisitos externos como sendo requisitos não funcionais, além dos requisitos de produtos. A figura exibe um conjunto de 7 requisitos não funcionais, sendo alguns destes ainda decompostos.

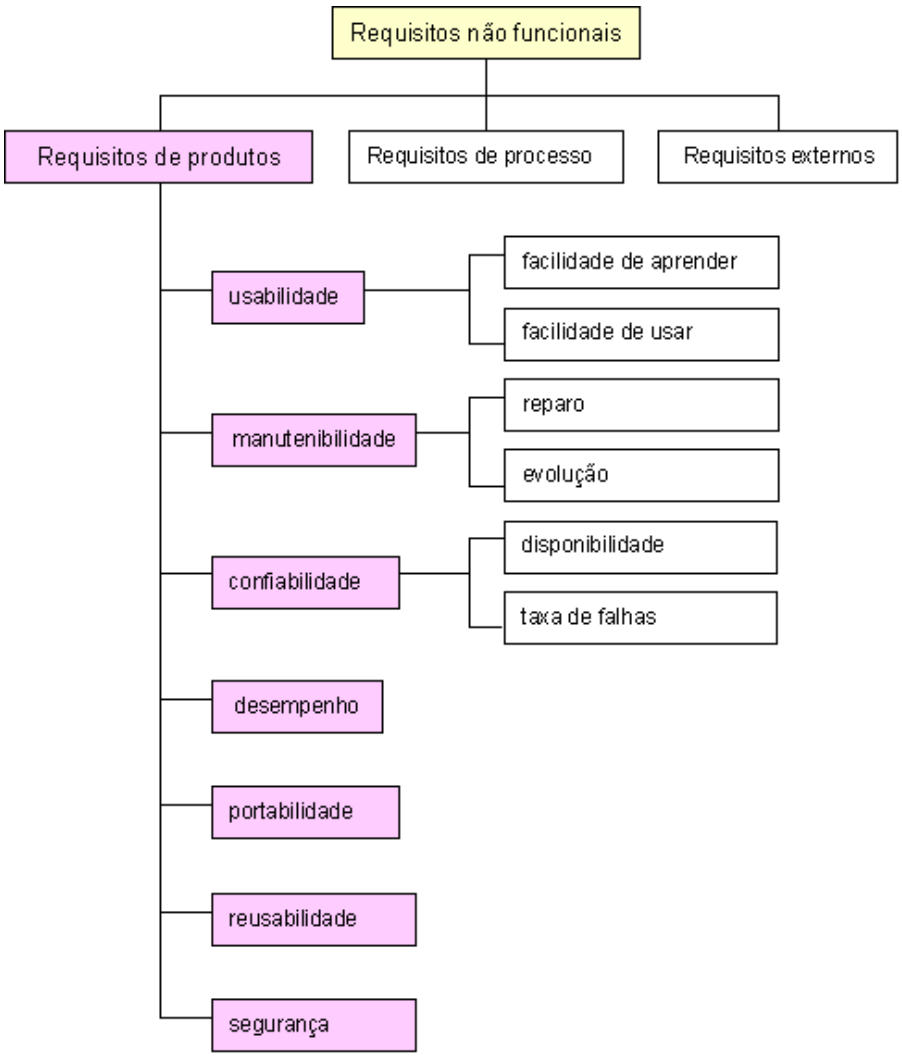

Figura 2. Tipos de requisitos não funcionais.

Usabilidade

Usabilidade é um dos atributos de qualidade ou requisitos não funcionais de qualquer sistema interativo, ou seja, no qual ocorre interação entre o sistema e seres humanos. A noção de usabilidade vem do fato que qualquer sistema projetado para ser utilizado pelas pessoas deveria ser fácil de aprender e fácil de usar, tornando assim fácil e agradável a realização de qualquer tarefa.

Requisitos de usabilidade especificam tanto o nível de desempenho quanto a satisfação do usuário no uso do sistema. Dessa forma, a usabilidade pode ser expressa em termos de:

- **Facilidade de aprender**: Associado ao tempo e esforço mínimo exigido para alcançar um determinado nível de desempenho no uso do sistema.
- **Facilidade de uso**: Relacionado à velocidade de execução de tarefas e à redução de erros no uso do sistema.

Os requisitos de usabilidade são coletados juntamente com outros requisitos (de dados e funcionais) usando algumas das técnicas de elicitação de requisitos como entrevistas ou observação. A coleta desses dados pode ocorrer, por exemplo, verificando o log de ações do usuário quando do uso de funcionalidade do sistema.

Esses requisitos de usabilidade podem ser expressos através de métricas de usabilidade, expressas em termos de medidas de desempenho. Tyldesley apresentou um conjunto de critérios que podem ser utilizados durante a medição de usabilidade [Tyldesley 1988]. A seleção de critérios a serem usados para mensurar a usabilidade depende do tipo de sistema. Exemplos de critérios de medição de usabilidade são apresentados na **Figura 3**.

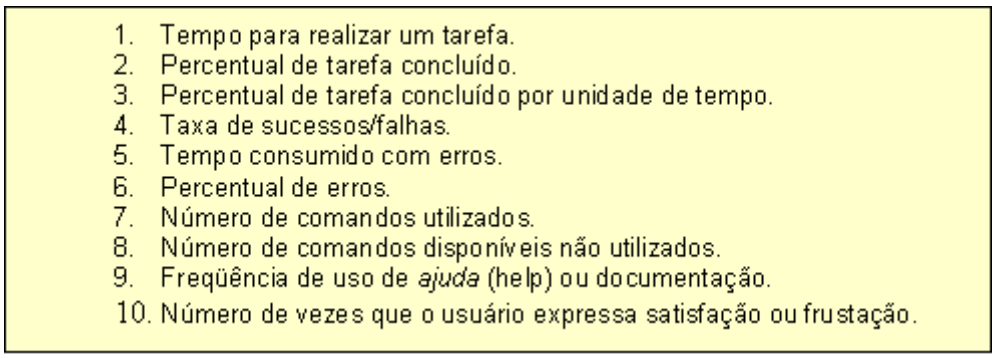
1. Tempo para realizar um tarefa.
2. Percentual de tarefa concluído.
3. Percentual de tarefa concluído por unidade de tempo.
4. Taxa de sucessos/falhas.
5. Tempo consumido com erros.
6. Percentual de erros.
7. Número de comandos utilizados.
8. Número de comandos disponíveis não utilizados.
9. Freqüência de uso de *ajuda* (help) ou documentação.
10. Número de vezes que o usuário expressa satisfação ou frustação.

Figura 3. Critérios de medição de usabilidade.

A definição das metas de usabilidade através de métricas é parte do processo denominado de engenharia de usabilidade. Neste processo, faz-se necessário também estabelecer os níveis desejados de usabilidade. Se, por exemplo, o usuário tem dificuldade em encontrar a funcionalidade desejada no sistema e, conseqüentemente, precisa recorrer à ajuda (Help) ou expressa insatisfação, então se observa que dois dos critérios da Figura 3 são considerados. A quantidade de vezes que essas ocorrências são observadas serve de indicador do suporte oferecido à usabilidade pelo sistema.

A usabilidade é um dos atributos de qualidade de um sistema e tem sido cada vez mais levada em consideração durante o desenvolvimento de software. A usabilidade pode ser afetada pelos componentes funcional (ou de aplicação) e de apresentação de um sistema. Mesmo que esses componentes sejam bem projetados, ainda assim a usabilidade poderá ficar comprometida se a arquitetura do sistema não levar em consideração a facilidade de efetuar uma modificação.

É importante acrescentar que a simples separação arquitetural entre componente de aplicação e apresentação em sistemas interativos não é suficiente para assegurar a usabilidade do mesmo. Assim, para determinar se uma arquitetura de software satisfaz a aspectos de usabilidade, torna-se necessário desenvolver cenários de uso do sistema. Em tais cenários, busca-se assegurar que informações corretas estejam disponíveis ao usuário no momento adequado, bem como encaminhar corretamente as instruções/comandos dos usuários a componentes apropriados do sistema. Note que a arquitetura de software do sistema tem um papel de suma importância visto que tais informações serão trocadas entre componentes do sistema e entre estes e o usuário, fluindo através de conectores da arquitetura.

Manutenibilidade

O termo manutenção de software é geralmente empregado quando nos referimos às modificações feitas após o sistema de software ter sido disponibilizado para uso. Na realidade, o termo manutenibilidade é um tanto abrangente já que ele envolve tanto a atividade de reparo (de algum defeito existente no sistema de software) quanto a atividade de alteração/evolução de

características existentes ou adição de novas funcionalidades não previstas ou capturadas no projeto inicial.

O reparo de um sistema de software ocorre quando defeitos são detectados, fazendo-se necessária a correção deles. A capacidade de efetuar um reparo depende do número de componentes do sistema. Por exemplo, se o sistema é monolítico, ou seja, constituído de um único componente, então tornar-se mais difícil efetuar o reparo se este sistema de software é de grande porte.

No entanto, se o sistema de software é modularizado, então tende a ser mais fácil analisar e reparar o existente. Podemos dizer que a modularidade favorece a capacidade de fazer reparo, permitindo que defeitos fiquem confinados a poucos módulos, considerando-se que se tenha a funcionalidade adequadamente separada. Uma observação importante é que a necessidade de fazer reparo é minimizada à medida que a confiabilidade do sistema aumenta.

Similarmente a outros sistemas, os sistemas de software evoluem ao longo do tempo, seja com a adição de novas funcionalidades ou com a modificação das existentes. Esta capacidade de evolução de um sistema de software é também influenciada pela modularidade do sistema. Note que se a arquitetura do sistema não considerar sua possibilidade de evolução por meio da adição e/ou alteração de funcionalidades do sistema, modificações tornar-se-ão cada vez mais difíceis à medida que o software evolui.

De um modo geral, a manutenibilidade é um dos requisitos mais relacionados com a arquitetura de um sistema de software. A facilidade de fazer alteração no sistema existente, seja adicionando ou modificando alguma funcionalidade, depende muito da arquitetura deste. Se considerarmos a necessidade de incrementar a quantidade de componentes encarregados do processamento de uma ligação telefônica (numa central telefônica) ou de transações eletrônicas, então esta adição de novos componentes deve ocorrer sem a necessidade de modificar a arquitetura existente e ainda comprometer pouco (ou nada) o desempenho atual do sistema. Tal suporte à manutenibilidade (seja para correção ou evolução do sistema) deve ser facilmente acomodada pela arquitetura de software.

É importante lembrar que uma arquitetura de software define os componentes e conexões entre estes e, portanto, também definem sob que circunstâncias componentes ou conectores podem ser alterados. Dessa forma, uma arquitetura ou estilo arquitetural de um sistema de software deveria efetivamente acomodar as modificações que precisarem ser feitas tanto durante seu desenvolvimento quanto após o sistema entrar em operação.

Confiabilidade

Confiabilidade de software é a probabilidade de o software não causar uma falha num sistema durante um determinado período de tempo sob condições especificadas. A probabilidade é uma função da existência de defeitos no software. Assim, os estímulos recebidos por um sistema determinam a existência ou não de algum defeito. Em outras palavras, a confiabilidade de software, geralmente definida em termos de comportamento estatístico, é a probabilidade de que

o software irá operar como desejado num intervalo de tempo conhecido. Também, a confiabilidade caracteriza-se um atributo de qualidade de software o qual implica que um sistema executará suas funções como esperado.

Os requisitos de confiabilidade compreendem restrições sobre o comportamento do sistema de software em tempo de execução. Na realidade, tem-se um conjunto de métricas de confiabilidade de software associadas a esses requisitos. Geralmente, as falhas de um componente de software são de natureza transitória, ou seja, elas ocorrem apenas para algumas entradas (estímulos) enquanto o sistema poderá continuar operando normalmente em outras circunstâncias. Isto distingue o software do hardware já que as falhas no segundo são de natureza permanente. Vale ressaltar que falha é o que se observa pelos usuários (externamente) enquanto que os defeitos, de origem interna ao sistema, são os motivadores das falhas.

Não é tão simples relacionar a disponibilidade de um sistema de software a uma falha existente, pois isto depende de vários fatores, tais como grau de corrupção de dados em decorrência de algum defeito de software e tempo de re-inicialização, dentre outros. Exemplos de métricas utilizadas para avaliar a confiabilidade de software compreendem:

- **Disponibilidade**: Esta é uma medida de quão disponível o sistema estaria para uso, isto é, quão disponível o sistema estaria para efetuar um serviço solicitado por algum usuário. Por exemplo, um serviço de um sistema de software terá uma disponibilidade de 999/1.000. Isto significa que dentre um conjunto de 1.000 solicitações de serviço, 999 deverão ser atendidas. Esta métrica é muito importante em sistemas de telecomunicações, por exemplo.
- **Taxa de ocorrência de falha**: Esta é uma medida da freqüência na qual o sistema falha em prover um serviço como esperado pelo usuário, ou seja, a freqüência na qual um comportamento inesperado é provável de ser observado. Por exemplo, se temos uma taxa de ocorrência de falha de 2/1.000, isto significa que 2 falhas são prováveis de acontecerem para cada 1.000 unidades de tempo.
- **Probabilidade de falha durante fase operacional**: Esta é uma medida da probabilidade que o sistema irá comporta-se de maneira inesperada quando em operação. Esta métrica é de suma importância em sistemas críticos que requerem uma operação contínua.
- **Tempo médio até a ocorrência de falha ou Mean Time To Failure (MTTF)**: Esta é uma medida do tempo entre falhas observadas. Note que esta métrica oferece um indicativo de quanto tempo o sistema permanecerá operacional antes que uma falha aconteça.

Qualquer métrica que venha a ser utilizada para avaliar a confiabilidade de um sistema dependerá da forma como o sistema é usado. Note ainda que o tempo é um fator considerado nas métricas. Ele é escolhido de acordo com a aplicação. Há sistemas de software que operam de forma contínua, enquanto outros operam de maneira periódica.

Por exemplo, considere um caixa eletrônico de banco. Este é um exemplo de sistema que opera periodicamente, isto é, um caixa eletrônico encontra-se parte do tempo em operação, enquanto no restante do tempo fica ocioso (embora disponível para uso por algum cliente do banco). No exemplo de um caixa eletrônico de banco, uma unidade de tempo mais adequada é o número de transações. Assim, um exemplo de falha seria a perda de dados entrados por um usuário. Neste

caso, a especificação de confiabilidade poderia ser a ocorrência de uma falha dessa natureza a cada 10.000 transações.

A arquitetura de software influenciará na confiabilidade de um sistema. O tempo médio até a ocorrência de uma falha ou MTTF poderá ser reduzido se houver replicação de componentes críticos. A perda de um desses componentes incorre em falha.

Uma forma de evitar isto ou contornar a perda de um componente é provendo uma arquitetura tolerante a falha onde uma réplica de um componente assume o processamento do componente com falha, evitando assim qualquer interrupção na operação de um sistema. Outra alternativa é degradar o desempenho do sistema sobrecarregando um componente com um número maior de requisições do que ele foi projetado. Dessa forma, a qualidade do sistema é degradada, mas ainda assim o sistema continua a funcionar (embora de modo precário até que uma ação corretiva seja tomada). A medida de confiabilidade pode ser definida em termos do tempo médio entre a ocorrência de falhas, ou Mean Time Between Failure (MTBF). Esta medida é dada por:

MTBF = MTTF + MTTR (MTTR ou Mean Time To Repair é o tempo médio de reparo)

A medida de disponibilidade também pode ser descrita em termo do MTTF e é definida como:

Disponibilidade = MTTF x 100% / (MTTF + MTTR)

Se considerarmos essas medidas, é importante observar que se o MTTR é reduzido, então a disponibilidade e confiabilidade do sistema serão maiores. Isto pode ser obtido arquiteturalmente se a separação de interesses for considerada durante o projeto. Perceba que quão menor é o tempo para proceder o reparo da falha, mais rapidamente o sistema voltará a ficar operacional e, portanto, a ficar disponível. Este atributo de projeto leva a uma maior integração, bem como maior facilidade nas modificações necessárias no sistema.

Note que adicionar componentes redundantes a um sistema de software implicará numa maior confiabilidade. Esta redundância é acrescentada na forma de verificações adicionais realizadas a fim de detectar erros antes que eles ocasionem falhas no sistema. Todavia, o uso de componentes redundantes acarreta numa redução de desempenho do sistema como discutido a seguir.

Desempenho

Desempenho é um atributo de qualidade importante para sistemas de software. Considere, por exemplo, um sistema de uma administradora de cartões de crédito. Em tal sistema, um projetista ou engenheiro de software poderia considerar os requisitos de desempenho para obter uma resposta de tempo para autorização de compras por cartão.

Note que os requisitos de desempenho têm impacto mais global sobre o sistema e, por essa razão, estão entre os requisitos não funcionais mais importantes. Contudo, é geralmente difícil lidar com os requisitos de desempenho e com outros requisitos não funcionais uma vez que eles estão em conflito, conforme discutido acima. No início da atividade de projeto da arquitetura de

software torna-se necessário definir quais requisitos não funcionais serão priorizados, dada a possibilidade de conflito entre eles.

Adicionalmente, desempenho é importante porque afeta a usabilidade de um sistema. Se um sistema de software é lento, ele certamente reduz a produtividade de seus usuários ao ponto de não atender às suas necessidades. Também, se o sistema de software requer muito espaço em disco para armazenamento de informações, pode ser oneroso utilizá-lo. Por exemplo, se um sistema de software exige muita memória para ser executado, ele pode afetar outras aplicações que são executadas no mesmo ambiente. Além disso, ele pode executar tão lentamente que o sistema operacional tenta balancear o uso de memória entre as diversas aplicações. De um modo geral, o requisito de desempenho pode ser decomposto em termos de tempo e espaço, como ilustrado na **Figura 4**.

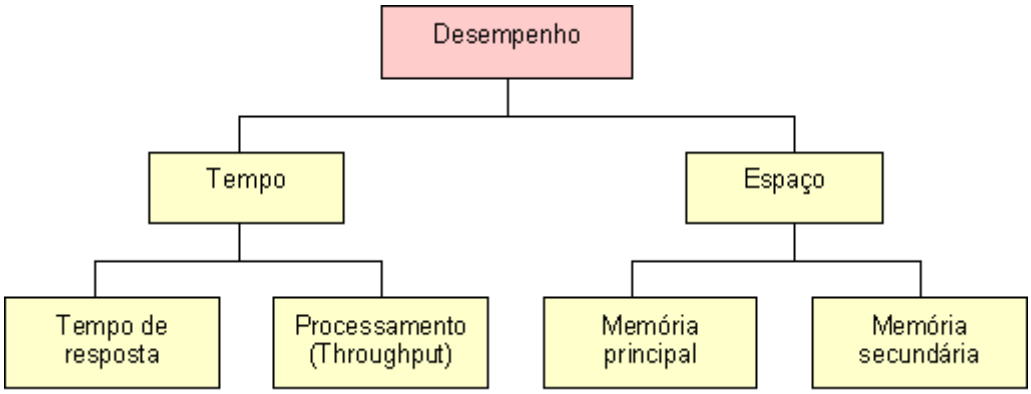

Figura 4. Fatores de desempenho.

O requisito de desempenho restringe a velocidade de operação de um sistema de software. Isto pode ser visto em termos de:

Requisitos de resposta

Especificam o tempo de resposta de um sistema de software aceitável para usuários. Neste caso, um projetista poderia especificar que o sistema deveria responder à solicitação de um serviço específico de um usuário dentro de um intervalo de 2 segundos. Por exemplo, num caixa eletrônico, após o usuário inserir o cartão magnético do banco no local apropriado (leitor do equipamento), o sistema deveria exibir uma nova tela, num intervalo de 2 segundos, requerendo que o usuário digite sua senha de conta corrente. Numa outra situação, o usuário pode ser solicitado a digitar sua senha e não o faz dentro de um período de 20 segundos, quando então um timeout ocorre e o sistema retorna à tela inicial.

Requisitos de processamento (throughput)

Estes requisitos especificam a quantidade de dados que deveria ser processada num determinado período de tempo. Um exemplo seria exigir que o sistema de software possa processar, no mínimo, 6 transações por segundo.

Requisitos de temporização

Este tipo de requisito especifica quão rapidamente o sistema deveria coletar dados de entrada de sensores antes que outras leituras de dados de entrada, feitas posteriormente, sobrescrevam os dados anteriores. Assim, por exemplo, poderia ser especificado que o sistema deveria efetuar leitura de dados 5 vezes por segundo, como condição mínima.

Requisitos de espaço

Em alguns casos, os requisitos de espaço podem ser considerados. Aqui, podemos nos referir à memória principal ou secundária. Por exemplo, a memória principal para executar uma aplicação poderia ser considerada como um requisito de desempenho uma vez que ela está relacionada ao comportamento do sistema em tempo de execução.

É importante observar que o desempenho depende da interação existente entre os componentes de um sistema de software e, portanto, está muito associado à arquitetura. Neste caso, os mecanismos de comunicação utilizados pelos componentes de um sistema têm influência sobre o desempenho obtido. Conforme vimos anteriormente, desempenho está relacionado a outros requisitos não funcionais. Por exemplo, a confiabilidade melhora com o uso de componentes redundantes. Todavia, o desempenho fica bastante comprometido, implicando na sua redução.

Portabilidade

Portabilidade pode ser definida como a facilidade na qual o software pode ser transferido de um sistema computacional ou ambiente para outro. Em outras palavras, o software é dito portável se ele pode ser executado em ambientes distintos. Note que o termo ambiente pode referir-se tanto à plataforma de hardware quanto a um ambiente de software como, por exemplo, um sistema operacional específico.

De um modo geral, a portabilidade refere-se à habilidade de executar um sistema em diferentes plataformas. É importante observar que à medida que aumenta a razão de custos entre software e hardware, a portabilidade torna-se cada vez mais importante. Adicionalmente, podemos ter a portabilidade de componentes e a portabilidade de sistemas. Esta última situação pode ser vista como caso especial de reusabilidade. O reuso de software ocorre quando todo o sistema de software é reutilizado, implementando-o em diferentes sistemas computacionais.

A portabilidade de um componente ou sistema de software é proporcional à quantidade de esforço necessário para que funcione num novo ambiente. Se uma quantidade menor de esforço é exigida quando comparada ao trabalho de desenvolvimento, então o sistema é dito portável.

Dois aspectos relevantes na portabilidade de programas são a transferência e adaptação. A transferência é o movimento de componente (código de um programa e dados associados) de um ambiente para outro. A adaptação engloba as modificações exigidas para fazer com que o programa seja executado em um novo ambiente.

Cabe salientar que o ambiente no qual um sistema de software opera é normalmente composto de hardware, sistema operacional, sistema de entrada e saída (E/S), bem como a aplicação. Uma

abordagem geral que poderia ser adotada para obter um sistema portável tentaria separar as partes do sistema que dependem do ambiente externo numa camada ou interface de portabilidade, conforme ilustrado na **Figura 5**.

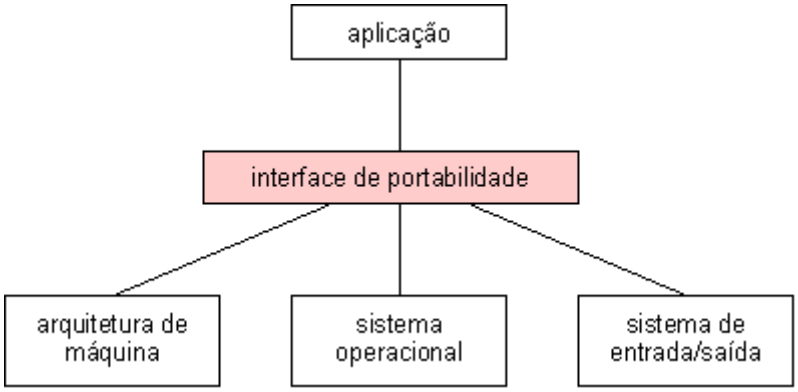

Figura 5. Separação de partes de um sistema computacional.

A interface de portabilidade mostrada na figura acima poderia ser vislumbrada e projetada como um conjunto de tipos de dados abstratos ou objetos, os quais iriam encapsular os elementos não portáveis procurando esconder características do software da aplicação. Assim, quando o sistema de software muda de hardware ou sistema operacional, apenas a interface de portabilidade precisaria ser alterada. Alguns problemas de portabilidade surgem, geralmente, devido à adoção de diferentes convenções para representar informação em arquiteturas de máquinas distintas, ou seja, hardware diferente.

Reusabilidade

Uma característica das engenharias é fazer uso de projetos existentes a fim de reutilizar componentes já desenvolvidos, objetivando minimizar o esforço em novos projetos. Dessa forma, componentes que já tenham sido desenvolvidos e testados podem ser reutilizados. Considere os elevados níveis de reusabilidade que encontramos tanto na indústria de automóveis quanto de aparelhos eletrônicos. Na indústria de automóveis, por exemplo, um motor é geralmente reutilizado de um modelo de carro para outro.

Em Engenharia de Software, à medida que aumenta a pressão para reduzir custos de desenvolvimento e manutenção de sistemas de software, bem como pela obtenção de sistemas com qualidade elevada, torna-se necessário considerar a reusabilidade como requisito não funcional no desenvolvimento de novos sistemas.

O reuso pode ser visto sob diferentes perspectivas. Ele pode ser orientado a componentes, orientado a processos ou orientado ao conhecimento específico de um domínio. Note que ainda poderíamos considerar o reuso de requisitos. Entretanto, aqui, iremos concentrar nossa atenção no reuso orientado a componentes. Exemplos desse tipo de reuso são:

- **Aplicação**: Toda a aplicação poderia ser reutilizada.
- **Subsistemas**: Os principais subsistemas de uma aplicação poderiam ser reutilizados.
- **Objetos ou módulos**: Componentes de um sistema, englobando um conjunto de funções, podem ser reutilizados.

- **Funções**: Componentes de software que implementam uma única função (como uma função matemática) podem ser reutilizados.

Dois tipos de reuso que estamos mais interessados são o reuso de subsistemas e objetos ou componentes, que chamaremos, simplesmente, de reuso de componentes. Este tipo de reuso não envolve apenas o código, mas também engloba a arquitetura e projeto associados.

É importante observar que podemos obter ganhos se reutilizarmos tanto projetos quanto arquiteturas. Isto minimiza esforços de desenvolvimento e requer menos alterações ou adaptações. Na realidade, quando se tem em mente que é necessário prover suporte à fácil modificação, indiretamente, obtemos componentes reutilizáveis.

Assim, o requisito reusabilidade pode envolver a arquitetura de um sistema de software ou componentes deste. O que determinará quão fácil será conseguir componentes reutilizáveis é a interdependência ou acoplamento entre os componentes. Por exemplo, uma biblioteca de componentes que oferece um conjunto de funcionalidades já implementadas e testadas oferece ao usuário (programador) um recurso valioso, já que basta incorporar esses componentes ao seu código e acessar suas funcionalidades através de sua interface.

Segurança

Em um sistema de software, este requisito não funcional caracteriza a segurança de que acessos não autorizados ao sistema e dados associados não serão permitidos. Portanto, é assegurada a integridade do sistema quanto a ataques intencionais ou acidentes. Dessa forma, a segurança é vista como a probabilidade de que a ameaça de algum tipo será repelida.

Adicionalmente, à medida que os sistemas de software tornam-se distribuídos e conectados a redes externas, os requisitos de segurança vão se tornando cada vez mais importantes. Exemplos de requisitos de segurança são:

- Apenas pessoas que tenham sido autenticadas por um componente de controle acesso e autenticação poderão visualizar informações dado que a confidencialidade permite esse tipo de acesso apenas às pessoas autorizadas.
- As permissões de acesso ao sistema podem ser alteradas apenas pelo administrador de sistemas.
- Deve ser feito cópias (backup) de todos os dados do sistema a cada 24 horas e estas cópias devem ser guardadas em um local seguro, sendo preferencialmente num local diferente de onde se encontra o sistema.
- Todas as comunicações externas entre o servidor de dados do sistema e clientes devem ser criptografadas.

Requisitos de segurança são essenciais em sistemas críticos, tais como sistemas de controle de vôo de aeronaves, uma vez que é impossível confiar num sistema deste tipo se ele não for seguro. Assim, pode-se dizer que todos os sistemas críticos possuem associados a eles requisitos de segurança. Os requisitos não funcionais de segurança envolvem diferentes aspectos. A **Figura 6** mostra tipos de segurança que podemos encontrar.

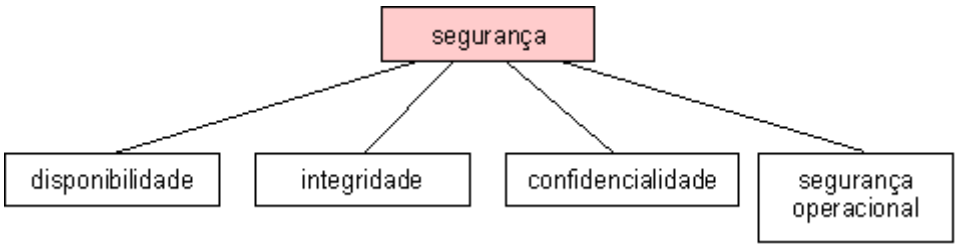

Figura 6. Tipos de segurança.

Também é importante considerar as diferentes ênfases encontradas para segurança:

- **Disponibilidade**: Refere-se a assegurar o sistema contra qualquer interrupção de serviço.
- **Integridade**: O foco na integridade ocorre principalmente em sistemas comerciais, onde se busca assegurar que acesso ou atualizações não autorizadas ocorram.
- **Confidencialidade**: A ênfase aqui é a de não permitir a revelação não autorizada de informações.
- **Segurança operacional**: Refere-se à fase considerada para o sistema em uso.

Note que para satisfazer ao requisito de qualidade não funcional de segurança em um sistema de software, alguns métodos podem ser empregados. Esses métodos podem ser vistos como um refinamento da meta de prover segurança a um sistema de software. Como exemplos desses métodos, podemos considerar:

Identificação

Identifica o nome do usuário, informando ao sistema quem está utilizando-o.

Autenticação

Visa assegurar que os usuários são, de fato, quem afirmam ser. Para tanto, fazem um teste de identidade. Este método envolve alguns aspectos, tais como:

- *Tipo de protocolo usado*: isto requer operação de senha.
- *Quantidade de autenticações*: pode requerer uma única senha ou múltiplas senhas ou procedimentos. Por exemplo, alguns bancos já fazem uso de múltiplas senhas durante operação de autenticação.
- *Partes envolvidas*: isto pode envolver a autenticação de uma parte envolvida (cliente) ou de ambas as partes envolvidas (cliente e sistema).

Tempo de acesso

Busca limitar o tempo de acesso ao sistema a fim de reduzir qualquer tipo de ameaça.

Auditoria de segurança

Objetiva habilitar pessoal autorizado a monitorar o sistema e, seletivamente, rastrear eventos importantes.

Alarme

Esta operação visa prevenir acessos, potencialmente suspeitos às informações vitais ou dados do sistema, notificando esses acessos à supervisão de segurança do sistema ou devidas autoridades.

A **Figura 7** apresenta alguns métodos que podem ser empregados em requisitos de segurança.

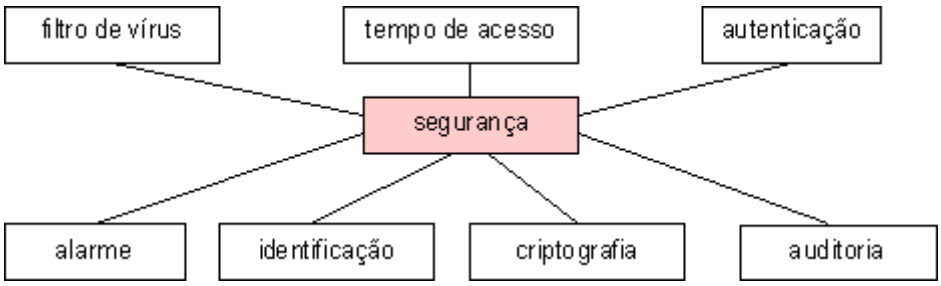

Figura 7. Métodos usados para prover segurança

É importante observar que a arquitetura de um sistema de software deve levar em consideração esses aspectos a fim de atender aos requisitos de segurança. Entretanto, o tipo de sistema determinará quais fatores precisarão ser levados em conta. Assim, poderá haver a inserção de componentes específicos de segurança bem como a conexão destes com outros componentes funcionais.

Conclusão sobre Requisitos Não Funcionais

A **elicitação de requisitos funcionais e não funcionais** é etapa fundamental no desenvolvimento de sistemas de software. Elementos de entrada desse processo de elicitação compreendem os principais influenciadores do sistema, envolvendo projetista, arquiteto e usuários.

Aliado a isso, a experiência do arquiteto de software é de grande importância. Como saída deste processo, tem-se um conjunto de requisitos funcionais oferecendo suporte às funcionalidades do sistema e uma lista de **requisitos não funcionais** oferecendo suporte à arquitetura de software. Cabe destacar que, quando da análise de arquiteturas candidatas para um sistema de software, um arquiteto ou engenheiro de software **considera os requisitos não funcionais como um dos principais critérios para sua análise**.

Por fim, é importante notar que uma cobertura completa de todos os **possíveis requisitos não funcionais** está fora do objetivo deste artigo. Para tanto, o leitor pode consultar o livro intitulado NonFunctional Requirements in Software Engineering de L. Chung, E. Yu, and J. Mylopoulos. Todavia, um subconjunto dos mais **proeminentes requisitos não funcionais** foi apresentado. Esses requisitos servem, via de regra, como critério para análise arquitetural objetivando a definição da arquitetura de software de um sistema.

Links Úteis

- Você conhece o método chinês?:

  O método chinês é uma técnica de depuração manual, amplamente utilizada no meio acadêmico, mas especialmente útil quando precisamos analisar um código sem ter o apoio de um editor.

- Criando meu primeiro projeto no Java:

  Neste curso você aprenderá a criar o seu primeiro programa com Java, e não, ele não será um simples “Hello, World!”.

- Delphi Exceptions: Trabalhando com exceções em Delphi:

  Neste curso você aprenderá a realizar o tratamento de exceções no Delphi, técnica que visa garantir o bom funcionamento da aplicação mesmo na ocorrência de certos erros.

Saiba mais sobre Engenharia de Software ;)

- O que é TDD?:

  Test Driven Development é uma técnica de desenvolvimento de software muito utilizada, por possuir algumas boas vantagens.

- Guias Engenharia de Software:

  Encontre aqui os Guias de estudo sobre os principais temas da Engenharia de Software. De metodologias ágeis a testes, de requisitos a gestão de projetos!

- Gestão de Projeto:

  Neste guia você encontrará o conteúdo que precisa para saber como gerenciar projetos de software. Confira abaixo a sequência de posts que te guiarão do básico ao avançado em Gestão de Projetos.

Artigo da Revista Engenharia de Software 3.

Referências:

Non-Functional Requirements in Software Engineering
http://www.utdallas.edu/~chung/BOOK/book.html

Are All Quality Goals Created Equal? Functional vs. Non-Functional
www.sei.cmu.edu/architecture/saturn/2005/quality_steven.pdf

Acquisition Practices: Good and Bad
www.sei.cmu.edu/programs/acquisition-support/conf/2003-presentations/oberndorf.pdf

SEI’s Software Architecture Technology Initiative
www.sei.cmu.edu/architecture/sat_init.html

The Software Architecture Portal

http://www.softwarearchitectureportal.org/