Desenvolvimento de Software requer Processo e Gestão

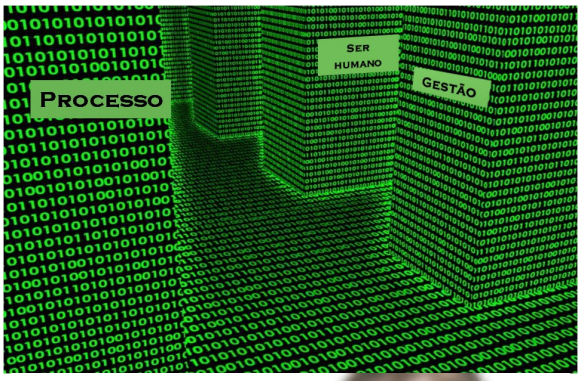
Antonio Mendes da Silva Filho*

"If Edison had a needle to find in a haystack, he would proceed at once with the diligence of the bee to examine straw after straw until he found the object of his search... I was a sorry witness of such doings, knowing that a little theory and calculation would have saved him ninety per cent of his labor."
Nikola Tesla

Software, de modo genérico, é uma entidade que se encontra em quase constante estado de mudança. Essas mudanças ocorrem por necessidade de corrigir erros existentes no software e/ou de adicionar novos recursos e funcionalidades. Igualmente, os sistemas computacionais (isto é, aqueles que têm software como um de seus principais elementos) também sofrem mudanças freqüentemente. Essa necessidade evolutiva do sistema de software o torna predisposto a defeitos (isto é 'não confiável'), podendo resultar em atraso na entrega e em custos acima do estimado. Concomitante com esses fatos, o crescimento em tamanho e complexidade dos sistemas de software exige que os profissionais da área raciocinem, projetem, codifiquem e se comuniquem por meio de componentes (de software) e qualquer concepção ou solução de sistema passa então para o nível arquitetural. Nesse sentido, o objetivo deste artigo é discutir e explorar o fato de que software não é construído como uma TV ou geladeira, mas desenvolvido. E, o desenvolvimento de software requer processo (de desenvolvimento) e gestão (do projeto de desenvolvimento). O artigo destaca a importância de se ter processo e gestão de projeto para obtenção de um produto ou serviço [1], [2], [3], [4] e [5].[1]

* **ANTONIO MENDES DA SILVA FILHO** é Professor e consultor em área de tecnologia da informação e comunicação, é autor dos livros Arquitetura de Software e Programando com XML, ambos pela Editora Campus/Elseviere e Doutor em Ciência da Computação pela Universidade Federal de Pernambuco.

[1] [1] Liderança, compromisso, confiança e plano de projeto: ingredientes essenciais à gestão de projetos, disponível http://periodicos.uem.br/ojs/index.php/EspacoAcademico/article/viewFile/13268/6972
[2] Intelecto Humano: Liderança Requer Compromisso e Compleição, disponível em http://www.periodicos.uem.br/ojs/index.php/EspacoAcademico/article/view/13040/6859



Quase cinco décadas atrás software constituía uma insignificante parte dos sistemas existentes e seu custo de desenvolvimento e manutenção eram desprezíveis. Para perceber isso, basta olharmos para história da indústria do software (www.softwarehistory.org). Encontra-se o uso do software numa ampla variedade de aplicações tais como sistemas de manufatura, software científico, software embarcado, robótica e aplicações Web, dentre tantas. Paralelamente, observou-se o surgimento de várias técnicas de modelagem e projeto bem como de linguagens de programação. Perceba que o cenário existente, décadas atrás, mudou completamente.

Antigamente, os projetos de sistemas alocavam pequena parcela ao software. Os componentes de hardware, diferentemente, eram analisados e testados quase exaustivamente o que permitia a produção rápida de grandes quantidades de subsistemas e implicava em raros erros de projetos. Entretanto, a facilidade inicial de modificar o software, comparativamente ao hardware, motivou seu uso. A intensificação do uso do software numa larga variedade de aplicações o fez crescer em tamanho e complexidade. Isto tornou proibitivo analisar e testar exaustivamente, além de impactar no custo de manutenção.

---

[3] Gestão de Projetos: Estratégia Essencial às Corporações, disponível em http://www.espacoacademico.com.br/066/66amsf.htm
[4] Componente de Software: componentização no desenvolvimento de software, disponível em http://www.espacoacademico.com.br/087/87amsf.htm
[5] Arquitetura de Software – sobre a importância do reuso, disponível em http://www.espacoacademico.com.br/068/68amsf.htm

Perceba que à medida que tamanho e complexidade dos sistemas de software aumentam, o problema de projeto extrapola as estruturas de dados e algoritmos de computação. Ou seja, há necessidade considerar o projeto da arquitetura (ou estrutura geral) do sistema, exigindo também um processo de desenvolvimento de software.

O **ciclo de vida de um produto** compreende um conjunto de atividades que vai desde sua concepção até a entrega desse produto ao cliente, envolvendo ainda sua evolução e eventualmente retirada desse produto. Aqui, nosso interesse recai sobre o produto software e, para ele, o ciclo de vida é composto de várias atividades que contribuem para o desenvolvimento do software. Agora, quais atividades propriamente ditas você tem num ciclo de vida de software?

- Concepção do sistema (de software)
- Levantamento de requisitos (de software)
- Análise de requisitos (de software)
- Especificação de requisitos (de software)
- Projeto de (de software)
- Implementação ou codificação (de software)
- Testes de (de software)
- Implantação ou entrega do software
- Manutenção e/ou evolução (de software)

Essas atividades compõem o que se costuma denominar de ciclo de vida do software. É importante salientar que você encontrará as atividades ilustradas na figura acima em processo de desenvolvimento de software como o RUP (Rational Unified Process) discutido adiante. Note que essas atividades são necessárias a qualquer tipo de software. Mas, o que é feito em cada uma dessas tarefas?

1. **Levantamento de requisitos** – Essa é a primeira atividade no desenvolvimento de um software, isto é, você irá fazer o levantamento de requisitos do sistema a ser desenvolvido. Aqui, você procura entender o que o cliente quer em termos de funcionalidades para o software. Por exemplo, num caixa eletrônico, funcionalidades desejadas pelo cliente são consultar saldo, imprimir extrato, fazer saque, dentre outras. Essas funcionalidades ou requisitos funcionais devem então ser documentados, gerando a primeira versão (ou rascunho) do documento de requisitos.
2. **Análise de Requisitos** – Depois que um conjunto de requisitos foram levantados, você terá que analisá-los para checar se todos foram identificados (isto é, se são completos), se são consistentes e se não existe ambigüidades. Finalizada a análise, você irá refinar o rascunho do documento inicial de requisitos para produzir a especificação de requisitos.

3. **Projeto** – Com a especificação de requisitos em mãos, você está em condições de adicionar detalhes essenciais para a implementação. Nesta atividade, por exemplo, você especifica os tipos de dados que serão utilizados além de definir os mecanismos de acesso a esses dados e como eles serão manipulados (ou processados).
4. **Implementação ou Codificação** – Nesta atividade o programa ou código (ou seja, conjunto de instruções) é escrito, implementando as funcionalidades que você levantou logo no início do ciclo de vida do software.
5. **Testes** – Como o próprio nome já denomina, aqui você é encarregado de testar o programa com o objetivo de achar bugs ou defeitos. Em outras palavras, você está interessado em sistematicamente tentar 'quebrar o código' para checar se ele funciona adequadamente ou não. Se você encontrar algum defeito, então você deverá corrigir o defeito antes de entregar o produto (software) para o cliente.
6. **Manutenção** – A outra grande atividade num ciclo de vida é manutenção de software que pode ser corretiva, visando corrigir defeitos encontrados durante as atividades de testes ou resultante de reclamações de usuários do software. Outra motivação para manutenção é quando o cliente ou usuários do software identificam a necessidade de adicionar novas funcionalidades ou de alterar alguma já existente. Em tal situação, você terá que fazer a manutenção a fim de atender às solicitações do cliente.

Se você para pensar um pouco sobre essas atividades de desenvolvimento, então você poderá perceber que essas atividades serão encontradas no desenvolvimento de qualquer software. Por exemplo, não se pode fazer um projeto se você não tem em mãos o conjunto de funcionalidades desejadas. Outra situação: não se pode testar algo que ainda não foi implementado. Pensando um pouco mais, você vai notar que existem relacionamentos entre essas atividades os quais são definidos e controlados por um processo.

Processo de desenvolvimento de software – Dentro do contexto discutido acima, um aspecto importante no desenvolvimento de um sistema de software é o contínuo feedback durante o processo. A importância de ter um *feedback* (resposta e comentários) do cliente mais cedo no desenvolvimento implica na necessidade de fazer um protótipo. Além disso, a necessidade de melhor planejar o desenvolvimento, fazendo um balanceamento entre custos e benefícios. Isto requer uma avaliação preliminar se é viável ou não desenvolver o software desejado pelo cliente. Como conseqüência, você pode perceber que duas atividades importantes são planejamento e análise de riscos, que procuram exatamente responder a questão: é viável desenvolver esse software?

Note que tudo isso visa minimizar custos e assegurar que o desenvolvimento irá ocorrer da forma como planejada e dentro dos prazos propostos. Agora, você poderia ainda questionar: Por que tudo isso?

Se você for analisar cuidadosamente, você perceberá que um processo requer *feedback* (do cliente), pois isto reduz as chances de problemas como, por exemplo, do projeto precisar de alterações à medida que ele avança, além de dar maior visibilidade do sistema. Portanto, um processo (de desenvolvimento) de software é necessário, porque ele:

- Serve de guia para controlar as atividades de desenvolvimento do sistema;
- Aloca tarefas para desenvolvedores específicos;
- Especifica quais artefatos precisam ser desenvolvidos em cada uma das etapas de desenvolvimento;
- Oferece critérios para monitorar as atividades de um projeto.

Para atender essas necessidades de modo mais adequado, um processo como o RUP (Rational Unified Process) e customizações dele tem sido empregado em projetos de software. O RUP é considerado como um *framework* ou arcabouço que serve para gerar processos. O RUP é um framework porque ele é configurável, ou seja, você pode customizar ou especializar o processo para diversos tipos de sistemas. Além disso, o RUP como processo agrega os métodos empregados no desenvolvimento e também faz uso da linguagem UML (Unified Modeling Language) para modelagem do sistema (a ser desenvolvido). O RUP é um processo que define bem o conjunto de atividades a serem executadas, além de informar os responsáveis pela execução delas. Adicionalmente, o processo explicita a ordem de execução das tarefas e se existe dependências entre elas, informando quais os artefatos de entrada e saída de cada tarefa. A **Figura 1** mostra uma visão geral do RUP.

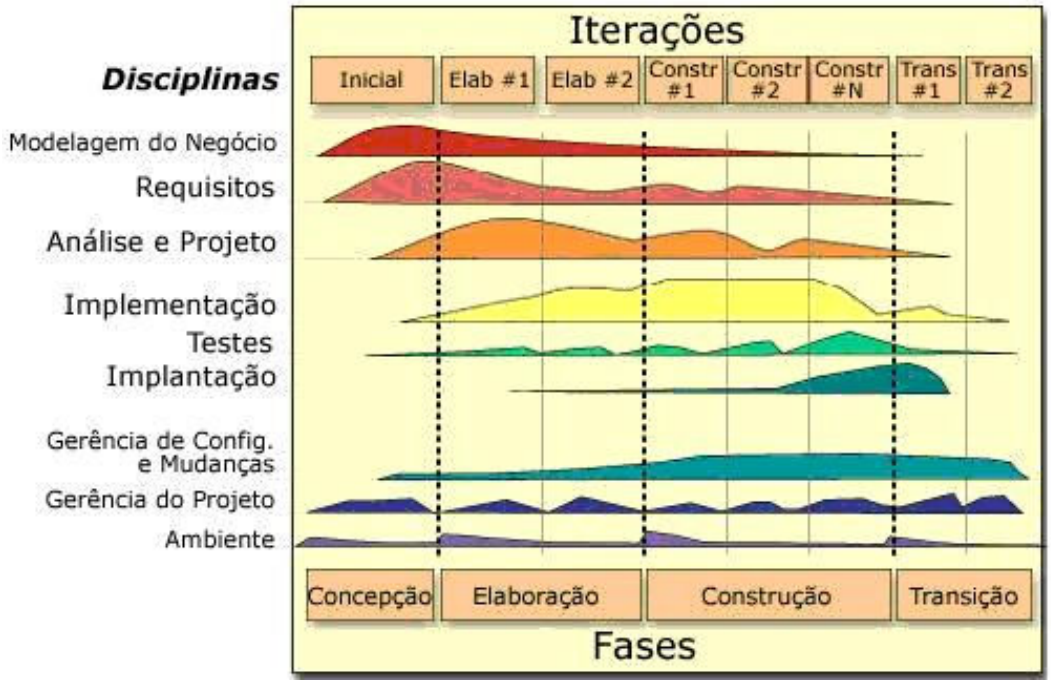


Figura 1 - Visão geral do RUP.

Fonte: www.wthreex.com/rup/portugues/index.htm acessado em Fevereiro de 2011

Observe a parte inferior da Figura 1, ela informa que o **RUP é composto de quatro fases**: concepção, elaboração, construção e transição. No lado esquerdo da figura, olhando verticalmente de cima para baixo, há um conjunto de disciplinas (como requisitos e implementação) que engloba atividades relacionadas. É importante destacar que essas disciplinas compreendem as atividades do ciclo de software, pois elas são necessárias ao desenvolvimento de um software. Outras disciplinas foram adicionadas como *gerenciamento de mudanças* e *gerenciamento de projeto*, as quais são essenciais no desenvolvimento de um sistema de software.

Como ressaltado acima, cada uma das disciplinas envolve várias atividades. Por exemplo, se considerarmos a disciplina *requisitos*, então nela o desenvolvedor deve analisar o problema de um cliente (ou seja, entender as funcionalidades que o cliente deseja para o sistema e quais as pessoas envolvidas) a fim de definir o escopo do sistema que consiste definir qual o conjunto de funcionalidades irá fazer parte do sistema. Observe que saber isso é muito importante, pois permitirá você trabalhar e gerenciar o escopo do sistema. O RUP determina como o sistema deve ser desenvolvido e, portanto, o desenvolvimento de sistema de software seguindo o processo RUP é:

- Interativo e incremental;
- Guiado por casos de uso (ou use cases);
- Centrado na arquitetura (de software).

Se você olhar na parte superior da Figura 1, então pode observar que o RUP é composto de iterações. Isto significa que todas as funcionalidades do sistema não precisam ser identificadas, especificadas, implementadas e testadas de uma única vez. O desenvolvimento se dá de modo incremental, ou seja, inicialmente, apenas as funcionalidades mais importantes são desenvolvidas e as demais são desenvolvidas em outras iterações, incrementando novas funcionalidades ao sistema. É como se cada iteração fosse um mini-projeto (no qual você teria de levantar e analisar requisitos, fazer o projeto, codificar e testar). Concluída parte do sistema, uma nova iteração (ou mini-projeto) seria feito até que todas as funcionalidades fossem implementadas. Note também que o RUP é guiado por casos de uso (ou use case).

Um caso de uso é um modelo que define uma funcionalidade do sistema sob a ótica de um ator, que pode ser um usuário, subsistema (de software) ou algum dispositivo de hardware (ou equipamento). Um ator, na grande maioria dos casos, será um usuário (humano) que interage com uma funcionalidade do sistema (descrita por um caso de uso). Um caso de uso descreve o que um usuário deve fazer para utilizar uma funcionalidade e como o sistema responde. Um exemplo de caso de uso é *sacar* num sistema de caixa eletrônico de um banco. Para sacar, o usuário precisa antes ter tido seu cartão do banco autenticado e no momento que o sistema (caixa eletrônico) solicita que o usuário informe a quantia que ele deseja sacar, o usuário então digita o valor de saque e aguarda a realização da operação com sucesso (caso o usuário tem saldo suficiente) ou não (se ele não possuir saldo). Nesse sentido, a especificação de um sistema de software compreende um conjunto de casos

de uso. Outras atividades de desenvolvimento como planejamento, análise, codificação e testes são baseadas e validadas com o modelo de caso de uso do sistema.

Além disso, o RUP é centrado na arquitetura o que significa dizer que o conjunto de funcionalidades vai ditar a forma na qual o sistema será desenvolvido e como poderá ter manutenção. Para entender isso, vamos fazer uma analogia com a planta baixa de uma casa que nos diz como os cômodos de uma casa estão distribuídos. Por exemplo, você tem sala, cozinha, banheiro e três quartos. Você olhando uma planta baixa ver como eles estão distribuídos e na hora de construir, você pode priorizar por construir a cozinha, banheiro e um quarto e, só depois, construir os outros 2 quartos. Em outras palavras, a planta baixa orienta o que tem de ser feito em termos de construção de uma casa e a arquitetura (de software) informa como ele deveria ser estruturado (ou organizado) e orienta como ele deveria ser desenvolvido.

Vale ressaltar que o RUP é uma evolução de propostas de modelo de desenvolvimento de software. Cabe também destacar que, ao mesmo tempo em que o RUP procura tratar questões pertinentes do desenvolvimento de software, o RUP pode ser customizado ou configurável para atender a necessidades especificas de um sistema de software.

De tudo o que foi apresentado anteriormente, ***a arquitetura de software (de um sistema) é determinante para adoção de um processo e gestão do projeto***. O modelo arquitetural de um (sistema de) software serve para definir a organização das funcionalidades deste sistema e propriedades ou requisitos não funcionais suportados pelo sistema. Características peculiares da arquitetura determinarão quais propriedades serão preponderantes. A necessidade da arquitetura de software prover suporte a um conjunto de requisitos, geralmente conflitantes, requer que uma análise arquitetural seja realizada.

Junto com a análise arquitetural, há o projeto da arquitetura de software que compreende uma etapa essencial no desenvolvimento de sistemas de software. Dentro deste contexto, a arquitetura de software é fundamental para o desenvolvimento de software onde se tem funcionalidades distintas sendo concebidas a partir da mesma arquitetura base. Entretanto, antecedendo a etapa de projeto da arquitetura de software, há a necessidade de fazer o levantamento dos requisitos do sistema como mostrado na Figura 2.

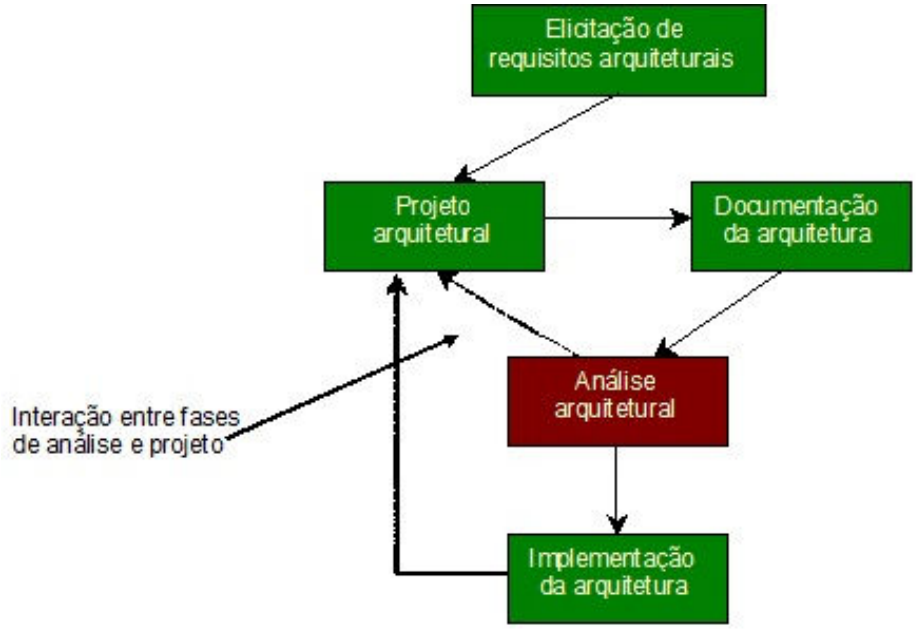


Figura 2 – Etapas no projeto da arquitetura de software.

De um modo geral, a análise e projeto de um sistema, geralmente, engloba as atividades de:

1. ***Desenvolvimento de modelos arquitetural*** – Os dados são coletados durante elicitação de requisitos a fim de serem analisados e incorporados ao modelo arquitetural (isto é, o produto resultante). Neste caso, o modelo passa a ser o elemento central da análise.
2. ***Melhoria e síntese de uma solução*** – Incrementa-se as informações descritas no modelo arquitetural (inicial).
3. ***Análise da solução*** – A análise da solução é realizada em termos do modelo arquitetural que se tem mãos, podendo identificar a necessidade de refinar o modelo.

Perceba que o modelo arquitetural pode ser considerado como um ‘esboço’ inicial da arquitetura de software do sistema em desenvolvimento. É importante ainda observar que a análise de uma arquitetura de software é um processo iterativo no qual são feitos refinamento de um modelo arquitetural inicial derivado a partir de um conjunto de requisitos arquiteturais. Em cada iteração, uma mini análise ocorre, refinando a arquitetura proposta inicialmente. Esse processo é ilustrado na Figura 3.

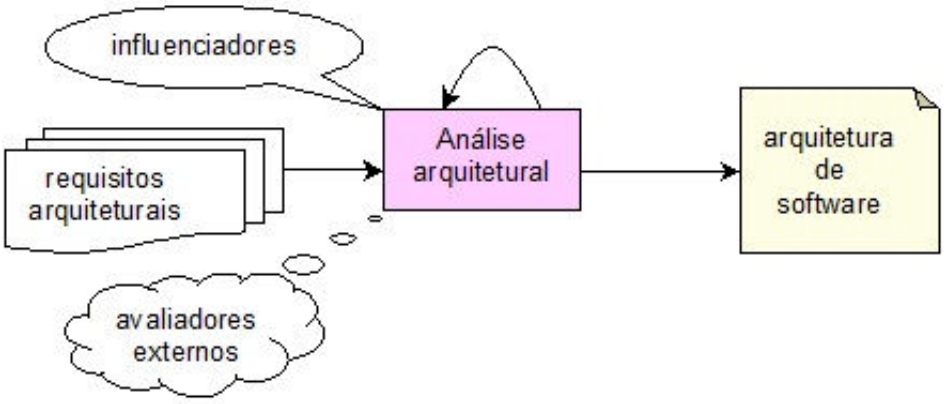


Figura 3 – Análise arquitetural.

Entretanto, note que não é tarefa fácil validar uma arquitetura de software quanto ao suporte que ela oferece a um conjunto de requisitos. É muitas vezes difícil associar novas arquiteturas a atributos de qualidade. Observa-se, geralmente, que os desenvolvedores concentram-se nos aspectos funcionais das arquiteturas. Assim, o principal propósito da análise arquitetural é verificar se os requisitos arquiteturais têm sido levados em conta durante o desenvolvimento de um sistema de software.

A arquitetura de software é uma parte essencial de um sistema de software complexo. Com a crescente complexidade de sistemas de software, a especificação global do sistema, ou seja, sua arquitetura passa a ser um aspecto de maior importância quando comparado à escolha de algoritmos ou estruturas de dados. Um sistema possui um conjunto de requisitos arquiteturais, envolvendo atributos de qualidade (ou requisitos não funcionais) e atributos de projeto. Esses requisitos constituem propriedades desejadas ou apresentadas por uma arquitetura de software. Dentre esses requisitos, uma arquitetura de software apresenta propriedades importantes que podem ser vistas como intrínsecas a ela. Tais propriedades são: *eficiência*, *integridade* e *flexibilidade*

A ***eficiência*** pode como a quantidade de recursos computacionais necessários para que um programa realize suas funções. Estes recursos computacionais podem ser vistos em termos da capacidade de armazenamento e processamento. Um dos principais requisitos associados à eficiência é o desempenho. O desempenho é um requisito não funcional essencial para um sistema de software e constitui-se num enorme desafio lidar com tal requisito durante o desenvolvimento de um sistema. Por exemplo, você poderia ter os seguintes requisitos de desempenho quando desenvolvendo um sistema de software para administração de cartões de crédito:

- Obter um bom tempo de resposta para autorização de vendas com cartão de crédito. Ao invés, essa especificação poderia ser re-escrita como obter um tempo de resposta para autorização de vendas com cartão de crédito de 7 ± 2 segundos.

- Obter um bom uso de memória para armazenar as informações de um cliente. Você poderia especificar o máximo de 10KB para armazenar as informações de cada cliente.

A eficiência é uma propriedade de suma importância nas decisões de projeto e tem forte influência na análise arquitetural de um sistema. Outra importante propriedade é a ***integridade***. Aqui, a noção de integridade refere-se a integridade em termos da unificação do projeto em níveis distintos. A arquitetura de software descreve a organização de um sistema de software num nível mais elevado, objetivando a unificação do projeto, ou seja, ela serve de referência para o projeto, conforme ilustrado na Figura 3.

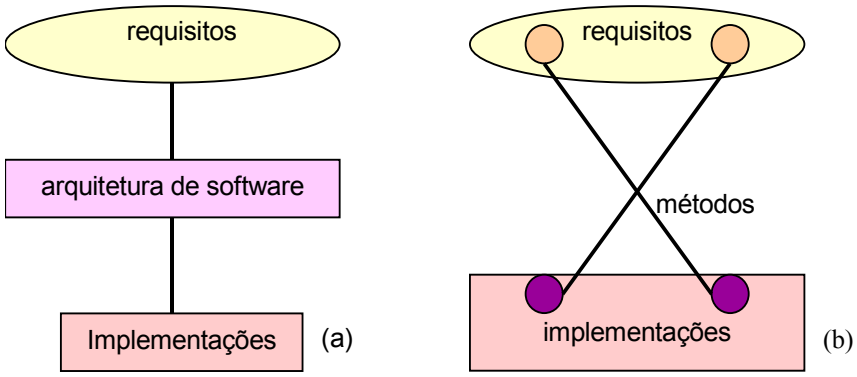


Figura 3 – Arquitetura de software como referência de projeto.

Note que a arquitetura de software é central e essencial à qualidade de um sistema em que a ela estão associados um conjunto de atributos de qualidade. Similarmente, a integridade (arquitetural) funciona como elemento coordenador em que ela unifica o projeto do sistema de software em diferentes níveis.

Outra propriedade importante é a ***flexibilidade*** que diz respeito ao esforço exigido para modificar um sistema de software. Em outras palavras, é a facilidade com a qual um sistema de software pode ser estendido a fim de dar suporte a uma ou mais funcionalidade(s) adicionada(s). Por exemplo, a criação de interfaces adicionais aumenta flexibilidade, incrementando a facilidade de fazer modificações, uma vez que as interfaces separam partes distintas de funcionalidade em componentes de software diferentes, como observado no estilo arquitetural orientado a objetos.

Adicionalmente, note que software é um produto (artefato) complexo, por natureza, exigindo atividades e processo bem definidos, além de execução disciplinada do projeto. **Software não é produzido** (no estrito sentido da palavra) como acontece com os carros, aparelhos de TV e outros equipamentos. **Diferentemente, software é desenvolvido**. Software requer um processo de desenvolvimento e não um processo de 'fabricação', implicando que ele exigirá o uso da engenharia sempre. Mas, por que software é considerado complexo?

Aqui, três fatores são destacados (sem a intenção de ser completo) como responsáveis por essa complexidade.

1. ***Software é um artefato de difícil compreensão e desenvolvimento.*** Freqüentemente, o engenheiro opta pelo software em detrimento do hardware na solução de problema por considerar sua manutenção mais fácil e também (em parte) pelo fato do software não 'sofrer' desgaste físico. Vale ressaltar que o (sistema de) software está sujeito a mudanças e evolução e, portanto, seu entendimento e desenvolvimento devem levar em consideração essa (futura) modificações. Outro fator impactante para compreensão e desenvolvimento de software é que ele não é um artefato ´visível' que seja facilmente e intuitivamente capturado por modelos 'tratáveis'. Isso é tudo? Não.
2. ***Software é desenvolvido e, portanto, requer que diversos artefatos sejam 'construídos'*** como, por exemplo, plano de projeto, documento de requisitos, modelo arquitetural, código, especificação de casos de teste, plano de teste, dentre outros. Observe que utilizar um processo de desenvolvimento requer diversos métodos para elicitação e análise de requisitos, método de projeto e implementação, métodos de teste e inspeção para cada dos artefatos gerados no desenvolvimento.
3. ***Software precisa satisfazer metas de produto, de projeto e de processo.*** Concomitante a esses métodos, você (engenheiro de software) faz uso da linguagem natural para elaborar documentos e de outras notações e linguagens especializadas para descrever modelos e escrever código. Não foi apenas isso, você ainda deve identificar os atributos de qualidade (ou requisitos não funcionais) que o software deve oferecer suporte. Atributos de qualidade como confiabilidade, eficiência, integridade, manutenibilidade, usabilidade e corretude, em geral, impõe dificuldade maior para o desenvolvimento. Dentro desse contexto, software deve satisfazer a metas de produto como eficiência e confiabilidade, deve atender metas de projeto (sendo executado em conformidade com cronograma e orçamento) e deve acomodar a metas de processo que antecipa possível evolução do produto.

Considerando os fatores destacados acima que contribuem para tornar software um artefato complexo, pode-se obter outra visão alternativa em termos de pessoas, processo e produto, como ilustrado no quadro abaixo.

Fatores influenciadores na complexidade de software	
Fator	***Exemplo***
Pessoas	Tamanho da equipe, nível de familiaridade com processo, nível de conhecimento, grau de motivação e envolvimento da equipe
Processo	Disponibilidade de ferramentas, disponibilidade e padrão de uso de métodos, ciclo de vida de projeto, restrições orçamentária e de cronograma, mecanismos de tratamento e rastreamento de mudanças de requisitos
Produto	Domínio da aplicação, grau de inovação do produto, atributos de qualidade exigidos (como interoperabilidade e confiabilidade)

E, tudo isso implica user experience. Todavia, há produtos para nichos específicos de usuários que requer especificidade na concepção e design, mas isso será explorado num outro artigo.