Projeto de Desenvolvimento de Software

Pedro Henrique Barboza da Silva

Aline Chagas Rodrigues Marques

Curso Técnico em Desenvolvimento de Sistemas

Educação a Distância

2019

Projeto de Desenvolvimento de Software

Pedro Henrique Barboza da Silva
Aline Chagas Rodrigues Marques

Curso Técnico em Informática

Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa

Educação a Distância

Recife

1.ed. | Nov. 2019

Licença Pública Creative Commons
Atribuição-NãoComercial-Compartilhalgual 4.0 Internacional

Professor(es) Autor(es)
Pedro Henrique Barboza da Silva
Aline Chagas Rodrigues Marques

Revisão
Pedro Henrique Barboza da Silva
Aline Chagas Rodrigues Marques

Coordenação de Curso
José Américo Teixeira de Barros

Coordenação Design Educacional
Deisiane Gomes Bazante

Design Educacional
Ana Cristina do Amaral e Silva Jaeger
Helisangela Maria Andrade Ferreira
Izabela Pereira Cavalcanti
Jailson Miranda
Roberto de Freitas Morais Sobrinho

Descrição de imagens
Sunnye Rose Carlos Gomes

Catalogação e Normalização
Hugo Cavalcanti (Crb-4 2129)

Diagramação
Jailson Miranda

Coordenação Executiva
George Bento Catunda
Renata Marques de Otero
Manoel Vanderley dos Santos Neto

Coordenação Geral
Maria de Araújo Medeiros Souza
Maria de Lourdes Cordeiro Marques

Secretaria Executiva de
Educação Integral e Profissional

Escola Técnica Estadual
Professor Antônio Carlos Gomes da Costa

Gerência de Educação a distância

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISDB

S586p

Silva, Pedro Henrique Barboza da.

Projeto de Desenvolvimento de Software: Curso Técnico em Desenvolvimento de Sistemas: Educação a distância / Pedro Henrique Barboza da Silva, Aline Chagas Rodrigues Marques. – Recife: Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa, 2019.

90 p.: il.

Inclui referências bibliográficas.

Caderno eletrônico produzido em outubro de 2019 pela Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa.

1. Informática – Desenvolvimento de software. 2. Qualidade de software. I. Título.

CDU – 004.41

Elaborado por Hugo Carlos Cavalcanti | CRB-4 2129

Sumário

Introdução

Engenheiros são responsáveis por criar soluções para os diversos problemas da humanidade nas mais diversas áreas, estes profissionais auxiliaram ao longo do tempo na evolução de tudo o que conhecemos. No desenvolvimento de sistemas não é diferente, para que soluções baseadas em sistemas de informação possam ser criadas, equipes de desenvolvimento de software se apoiam em princípios de engenharia aplicadas na construção de sistemas para realizar atividades tais como: elicitação de requisitos, modelagem de arquitetura, construção, testes e implantação. Nesta disciplina vamos aprender como estas atividades são organizadas em forma de processos, como estes processos evoluíram ao longo do tempo para atender necessidades decorrentes dos usuários de sistemas e as formas de avaliação da qualidade dos sistemas produzido e do processo utilizado.

Na primeira semana vamos entender o contexto da criação dos processos, a necessidade de um modelo de engenharia que permitisse a criação de software de forma sistemática com foco na qualidade. Em seguida vamos entender o que é um processo de desenvolvimento no ponto de vista genérico com atividades básicas que guiaram o desenvolvimento de software ao longo do tempo. Vamos observar a evolução dos modelos descritivos que privilegiam a descrição das atividades e a documentação de forma mais rígida, até os modelos ágeis que focam nas habilidades dos personagens envolvidos no processo de desenvolvimento, observando a modernização do processo, visando atender as necessidades do cliente de forma mais eficiente e produzindo um artefato (software) de maior qualidade com um custo viável.

Na segunda semana vamos entender os aspectos relacionados à qualidade, observando que qualidade em contextos de desenvolvimento de sistemas existem em dois aspectos: o produto em si e o processo. Vamos conhecer modelos de qualidade para ambas as áreas, entender o que cada modelo de qualidade busca avaliar, por fim conhecer as etapas de cada modelo.

Por fim esperamos contribuir com mais esse passo no processo de aprendizado no curso de Desenvolvimento de Sistemas, assistam as videoaulas, realizem as atividades propostas e em caso de dúvidas nos procure nos fóruns.

Bons estudos!

1.Competência 01 | Introdução a Projeto de Software

Nesta primeira semana vamos entender os contextos para a criação da engenharia de software como área de conhecimento responsável pela criação de software de forma sistemática, visando a entrega de um produto com qualidade, construído em um tempo razoável e com um custo adequado. Vamos estudar também a evolução desses modelos de avaliação e entender os aspectos que tornaram essas mudanças relevantes. Realize a leitura da competência, assista a vídeo aula, acesse o material complementar indicado, pois ele também é importante para a construção do conhecimento no nosso modelo de aprendizado, realize as atividades propostas e em caso de dúvida nos procure nos fóruns.

1.1 O que é Software?

Muitos associam software apenas aos programas de computador, isso seria uma definição muito restritiva, na verdade podemos definir software não apenas como o programa, mas também a tudo associado à sua implementação, os dados de documentação e as configurações necessárias para que o software funcione corretamente.

Para Sommerville (2011) um sistema de software consiste, geralmente, de um conjunto de programas separados; arquivos de configuração, que são utilizados para configurar esses programas; documentação de sistemas, que descreve a estrutura do sistema, documentação do usuário que explica como usar o sistema, e por meio dos quais os usuários obtém informações sobre o produto.

Pressman & Maxin (2016) definem software como o produto que profissionais desenvolvem e dão suporte a longo prazo. Abrange programas executáveis em um computador de qualquer porte ou arquitetura, conteúdos, informações descritas tanto na forma impressa quanto na virtual, abrangendo qualquer mídia eletrônica.

O que precisamos entender inicialmente é que um software não pode ser definido simplesmente como um programa de computador ou um aplicativo de smartphone, existem diversos processos e papéis envolvidos, desde a sua concepção até a entrega ao usuário final e o entendimento de todo esse processo, será o nosso objetivo nestas duas competências da nossa disciplina.

1.2 Engenharia de Software

A engenharia é o ramo das ciências responsável por criar soluções para as diversas necessidades do ser humano ao longo do tempo. Engenheiros já atuam há muito tempo, em distintas áreas do conhecimento, criando, inovando e desenvolvendo a tecnologia em soluções para construção civil, eletrônica, meio ambiente, telecomunicações e para a área de desenvolvimento de sistemas não seria diferente.

Nesse contexto temos a engenharia de software, como o ramo dos estudos científicos relacionados a padronização das atividades que permitem a criação de um software, seja um programa de computador, um sistema web, um aplicativo para smartphone ou um jogo.

Por ser mais nova que a maioria das engenharias, a engenharia de software, ainda passa por mudanças e inovações em seus processos, mais a frente trataremos dessas mudanças e evoluções. Por hora iremos focar no seu objetivo que é relacionar todos os aspectos envolvidos na produção do software, desde a sua concepção até a operação, de maneira eficaz.

A engenharia de software nasce com o aumento da demanda por software, por volta da década de 70, com a evolução do hardware e a necessidade de programas que solucionassem os problemas encontrado na época, o desenvolvimento de sistemas passou a ser uma atividade mais frequente e não se tinha um modelo de desenvolvimento definido. Nesse cenário alguns problemas passaram a surgir em relação a esta atividade, como lista Wazlawick (2013).

- Projetos que estouram o cronograma.
- Projetos que estouram o orçamento.
- Produto de baixa qualidade ou que não atenda aos requisitos.
- Produtos não gerenciáveis e difíceis de manter e evoluir.

Esse período ficou conhecido como crise do software, para diluir os problemas encontrados nesta época, surgem as primeiras versões dos modelos de produção de software. Visando criar software de maneira sistemática, com qualidade e em um tempo razoável.

Na figura 1 a seguir, Pressman & Maxin (2016) representam de maneira direta os conceitos sobre engenharia de software, como sendo a utilização de ferramentas, métodos e processo, para desenvolver software com foco na qualidade.

Para que essas demandas (sistemas confiáveis e custos reduzidos) sejam supridas, os autores propõem que a engenharia de software trabalhe através de uma visão em camadas, que os autores chamam de pilares da engenharia de software. Esses pilares são formados por quatro camadas, como mostra a figura 1.

Figura 01 - Engenharia de Software em Camadas
Fonte: Pressman & Maxin (2016).
Descrição: A figura apresenta quatro camadas circulares sobrepostas, com as escritas ferramentas, métodos, processos e foco na qualidade respectivamente. Representado os conceitos relacionados a engenharia de software, como sendo a utilização de ferramentas, métodos e processo, para desenvolver softwares com foco na qualidade, definido por Pressman & Maxin (2016).

Nas palavras dos autores esses pilares representam:

- Foco na qualidade: constitui a base do pilar e é a camada que guia todo o desenvolvimento de software, uma vez que a gestão da qualidade permite que se aperfeiçoe continuamente os processos.
- Processo: permite a racionalização do desenvolvimento de software, uma vez que é através dessa camada que se informa quais métodos serão implantados, como as ferramentas serão utilizadas e como a entrega do produto será feita. Além disso, define a metodologia que deverá ser conduzida para que se possa ter uma entrega efetiva do produto.
- Métodos: é a camada que especifica informações técnicas em como produzir o software. Os métodos possuem uma série de tarefas atreladas aos métodos de desenvolvimento, como: análise de requisitos, estimativas do projeto, teste, manutenção, entre outros.
- Ferramentas: propicia apoio parcial ou totalmente automatizado aos métodos. Quando as ferramentas são integradas, de modo que as informações criadas por uma

ferramenta possam ser usadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de software denominado CASE (Computer Aided Software Engineering ou Engenharia de Software Auxiliada por Computador).

Após compreender a funcionalidade da Engenharia de Software no processo de desenvolvimento de aplicações, a seguir veremos como os modelos de processo oriundos dessa engenharia funcionam.

O que faz um engenheiro de software??
https://www.youtube.com/watch?v=BfzrAIHGbDU

1.3 Processos de Desenvolvimento de Software

Com a necessidade de técnicas que acompanhassem a popularização e o progresso das aplicações, surgiram modelos prescritivos de processo com o objetivo de estruturar e guiar as tarefas relacionadas ao desenvolvimento de software (Pressman, et al., 2016).

Essas tarefas correspondem a:

- Comunicação: que deve acontecer entre desenvolvedores e clientes para que a informação possa fluir de forma adequada durante o desenvolvimento de software.
- Planejamento: que estrutura os processos e os prazos para a entrega do software.
- Modelagem: que consiste na forma como o software será construído.
- Construção: que está relacionada à codificação do software.
- Implantação que provê a entrega de uma versão final do software para o cliente.

Além disso, os modelos de processo apresentam um fluxo de trabalho, o que permite a padronização dos artefatos, auxiliando, a gerência do desenvolvimento de software (SOMMERVILLE, 2011).

Um processo de software é um conjunto de atividades relacionadas que levam à produção de um sistema qualquer. Como será visto adiante, existem alguns modelos de processo (metodologias) para o desenvolvimento de sistemas, porém existem atividades que podem ser identificadas dentro de cada modelo existente, segundo Sommerville (2011) são elas:

- **Especificação de software:** A funcionalidade do software e as restrições a seu funcionamento devem ser definidas.
- **Projeto e implementação de software:** O software deve ser produzido para atender às especificações.
- **Validação de software:** O software deve ser validado para garantir que atenda às demandas do cliente.
- **Evolução de software:** O software deve evoluir para atender às necessidades de mudança dos clientes

Como vários autores afirmam não existe um processo ideal, a maioria das empresas e equipes que produzem software desenvolve e adaptam os próprios processos de desenvolvimento de software para o que for adequado. Neste sentido os processos têm evoluído de maneira a tirarem melhor proveito das capacidades das pessoas em uma organização, bem como das características específicas do sistema em desenvolvimento. Para alguns sistemas, como sistemas críticos, é necessário um processo de desenvolvimento muito bem estruturado, a exemplo do modelo cascata; para sistemas de negócios, com requisitos que se alteram rapidamente, provavelmente será mais eficaz um processo menos formal e mais flexível, como os ágeis (Sommerville, 2011).

Aliados aos conceitos já vistos de processos, outros elementos são comumente encontrados nos estudos relacionados aos processos de software. Wazlawick (2013) define muito bem estes elementos, como veremos a seguir.

O primeiro é o conceito de **fase**. Uma fase é um período de tempo no qual determinadas atividades com objetivos bem específicos são realizadas. As fases são, então, as grandes divisões dos processos, normalmente sua quantidade é pequena. As fases podem ser sequenciais, ou seja, com o passar do tempo o processo de desenvolvimento passa de uma fase a outra, como requisitos, análise, programação, testes e implantação. Ou podem ser cíclicas, ou seja, o desenvolvimento passa

repetidamente de uma fase para outra, formando um ciclo repetitivo de fases até a finalização do projeto.

Outro termo encontrado é **disciplina**, que pode ser entendida como um conjunto de atividades ou tarefas correlacionadas, as quais servem a um objetivo específico dentro do processo de desenvolvimento. As disciplinas usualmente são compostas por tarefas ou atividades que se organizam em relação as suas dependências, que estabelece em que ordem (se for o caso) as atividades devem ser executadas. Existem, por exemplo, disciplinas relacionadas à produção, como análise de requisitos, modelagem, programação etc., mas também existem disciplinas de apoio, como gerência de projeto, ambiente e gerência de configuração.

O termo **tarefa** ou **atividade** pode ser associado aos processos de software, independente da metodologia que se utilize, pois, a maioria dos processos de software é organizada em torno de tarefas, às vezes também chamadas de atividades. Toda atividade tem um objetivo principal estabelecido e visa criar ou produzir uma mudança de estado visível em um ou mais artefatos durante a execução de um projeto.

Associados aos conceitos de tarefa, temos outros determinantes para o entendimento de sua realização. São eles:

- **Artefatos**: são quaisquer documentos que puderem ser produzidos durante um projeto de desenvolvimento de software, incluindo diagramas, programas, documentos de texto, desenhos, contratos, projetos, planos etc. Geralmente representam o resultado de uma atividade.
- **Responsáveis**: os responsáveis são perfis de pessoas ou cargos que respondem pela realização de uma atividade. Na prática, é interessante que qualquer atividade tenha um único responsável.
- **Recursos**: Uma atividade, para ser executada, pode demandar recursos. Existem **recursos humanos** e **recursos físicos.** Em geral, os recursos humanos são classificados à parte dos recursos físicos. Os **recursos humanos** são associados às atividades nos papéis de responsáveis e participantes. Os recursos físicos, de outro lado, dividem-se em dois grupos: consumíveis e não consumíveis.

- **Recursos consumíveis:** são aqueles que são gastos quando usados. Por exemplo, folhas de papel, passagens etc. Esses recursos normalmente são alocados em determinada quantidade.
- **Recursos não consumíveis**: podem ser alocados inúmeras vezes para várias atividades, porém, normalmente não podem ser alocados para mais de uma atividade de cada vez. Exemplos de recursos não consumíveis são o software e o hardware. Por exemplo, enquanto um computador estiver sendo usado em uma atividade, não poderá ser simultaneamente usado em outra. Contudo, depois de liberado, poderá ser usado novamente.

Uma vez entendidos os conceitos de processos de software, passaremos a alguns modelos de processos (metodologias), que nada mais são do que aplicações diversas dos conceitos de processos com características específicas, tratando de forma a atender as necessidades existentes no tocante a projetos de desenvolvimento de software ao longo do tempo.

1.4 Metodologias de Desenvolvimento de Sistemas

Um modelo de processo de software, ou uma metodologia de desenvolvimento de sistemas, por sua vez, é uma representação simplificada do processo, ou seja, os modelos demonstram uma visão particular de um processo ou de um conjunto de processos, fornecendo informações em representações genéricas do contexto a ser desenvolvido no projeto.

Em resumo, o modelo é um framework do processo sem o detalhamento de suas atividades específicas. Além disso, eles não são estáticos e podem ser utilizados para diferentes abordagens de desenvolvimento de software (Pressman, et al., 2016).

Para escolher um modelo de processo adequado é necessário obter uma visão holística do desenvolvimento de software. Em suma, um modelo preciso focar no prazo e na qualidade preestabelecida e estar de acordo com as necessidades específicas de cada projeto.

Após definirmos as diferenças existentes entre os processos e modelos de processos, a seguir, veremos alguns tipos de modelo existentes.

1.4.1 Modelo Cascata

Esse modelo foi introduzido, inicialmente, em 1970 por Winston W. Royce e corresponde a um modelo que segue uma sequência de tarefas.

O nome Cascata é devido à sua continuidade simples em que, em cada fase do modelo, tem-se as principais etapas de desenvolvimento de software, conforme a figura 2 a seguir ilustra:

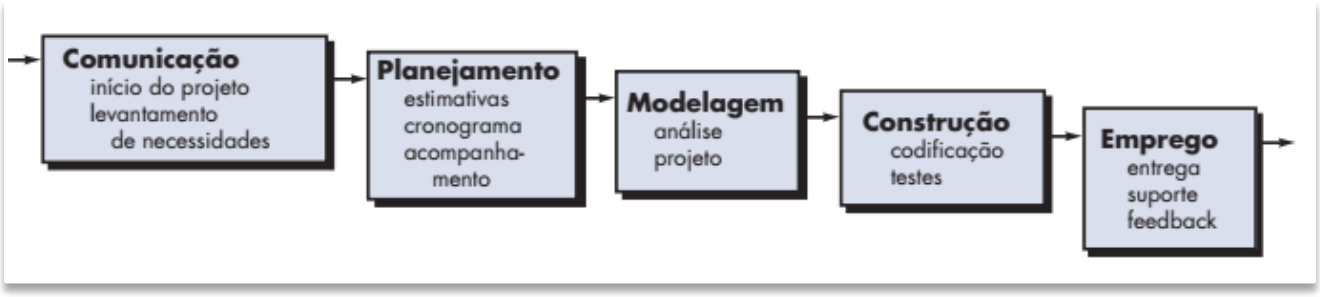

Figura 02 - Modelo Cascata
Fonte: (Pressman, et al., 2016).
Descrição: a figura apresenta 5 caixas alinhadas em sequência, representando as 5 atividades prescritas em cada etapa da modelo cascata: comunicação, planejamento, modelagem, construção e comunicação.

No modelo em cascata, uma etapa somente começa após a finalização e a conclusão da etapa anterior, seguindo, assim, uma ordem sequencial de desenvolvimento de projeto. Quando a Fase 1 é finalizada, inicia-se a Fase 2; quando a Fase 2 é encerrada, inicia-se a Fase 3, e assim por diante.

Em cada uma dessas fases, o modelo em cascata produz novos artefatos, tornando-se, assim, consistente. Dessa maneira, o processo pode ser visualizado pelos gerentes de forma ampla, permitindo que eles acompanhem se o progresso de desenvolvimento está em conformidade com o plano preestabelecido (Sommerville, 2011).

Desse modo, o desenvolvimento de software através do Modelo Cascata segue uma sequência de etapas ordenadas em que, ao final de cada fase, tem-se uma revisão (Hirama, 2012). Além disso, o processo não prossegue até que o cliente tenha a completa satisfação dos resultados apresentados pelos artefatos gerados em cada fase do desenvolvimento. Algumas características deste modelo são:

- As atividades de especificação, codificação e testes seguem uma disciplina.
- Uma atividade não é iniciada sem que a anterior tenha sido encerrada e aprovada.

- Há uma sequência rígida de atividades.
- O usuário/cliente é envolvido somente no início e no fim do processo.

O modelo em cascata é o paradigma mais antigo, mas é necessário entende-lo como um ponto de partida da engenharia de software e relacionar com as mudanças dos modelos mais modernos, a exemplo das metodologias ágeis. Ao longo das últimas três décadas, as críticas a este modelo de processo fizeram com que até mesmo seus mais ardentes defensores questionassem sua eficácia, por conta da rigidez na forma de trabalhar (Pressman, et al., 2016).

1.4.2 Modelos Incrementais e Evolucionários

Quando se tem a necessidade de oferecer um conjunto de funcionalidades ao cliente de forma rápida, podem ser empregados modelos de processo que tem por finalidade apresentar uma evolução do produto que está sendo construído. Nesses modelos, versões funcionais do software são entregues aos usuários, possibilitando, assim, que a equipe de desenvolvimento refine as funcionalidades antes da entrega final (Pressman, et al., 2016).

Desse modo, modelos incrementais e evolucionários são responsáveis por produzir uma série de versões, conhecidas como versões ou incrementos, que são compostos por um conjunto de funcionalidades. À medida que esses elementos são construídos, eles são avaliados pelo usuário, que através de *feedback*, vão refinando os resultados apresentados, até que seja apresentada uma versão adequada do sistema.

Em desenvolvimento de sistemas, protótipos podem ser entendidos como versões iniciais de um sistema, utilizados experimentar ideias e avaliar as possíveis soluções, geralmente com a ajuda de *stakeholders* que estão envolvidos no processo.

Wazlawick (2013), apresenta duas formas distintas de se trabalhar com protótipos:

- A primeira é denominada ***throw-away***, onde os protótipos são utilizados para estudar do sistema, auxiliando no entendimento dos requisitos e diminuindo os riscos. Após a finalização destas etapas deve ser descartado.

- A segunda forma de trabalhar com protótipos é denominada, ***cornestone***, que além de cumprir os mesmos objetivos do modelo anterior, deve ser utilizado como ponto de partida para a evolução do sistema.

O modelo de **prototipação evolucionária**, é baseado na ideia da prototipação *cornestone*, pois, é necessário um planejamento para a manutenção dos defeitos iniciais dos protótipos, para que não tenham impacto na versão final do sistema (Wazlawick, 2013). A utilização deste tipo de recurso, permite ao usuário visualizar versões parciais do sistema, bem como obter novos requisitos e apontar pontos de melhoria e reforçar pontos fortes do produto (Sommerville, 2011). A Figura 3 a seguir apresenta um modelo para o desenvolvimento baseado em protótipos.

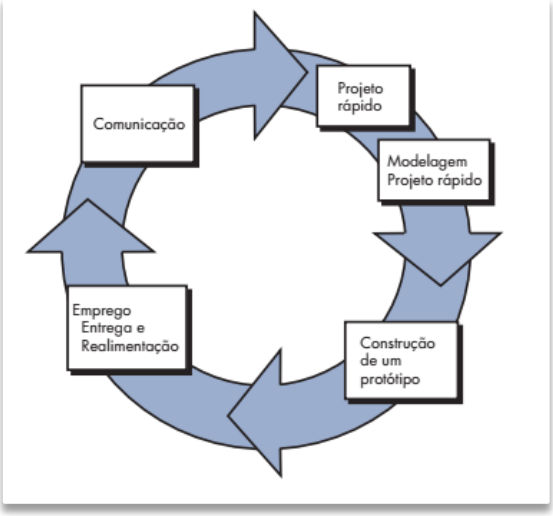

Figura 03 - Modelo de Processo Prototipação
Fonte: (Pressman, et al., 2016).
Descrição: A figura apresenta as atividades descritas para um processo de software na visão de Pressman contornadas em forma de círculo, o círculo por sua vez é formado por setas que dá uma ideia de continuidade, que o processo é realizado de forma contínua.

O modelo de prototipação sugere que sejam desenvolvidos aspectos visíveis do sistema, até que se produza uma versão aceitável do sistema. A Figura 04 apresenta um esquema para o desenvolvimento de protótipos. Esse processo pode ser utilizado para construção de um protótipo para uma interface de usuário, validar requisitos funcionais de um sistema ou validar a viabilidade de uma solução (Sommerville, 2011).

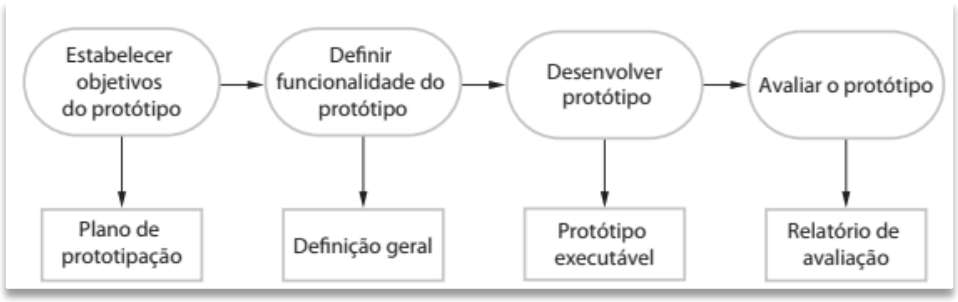

Figura 04- Processo de Criação de Protótipos
Fonte: (Sommerville, 2011).
Descrição: A figura apresenta na parte superior as 4 atividades na construção de um protótipo e em baixo as atividades relacionadas com cada etapa, setas ligam cada etapa da parte superior a cada atividade pertencente a ela, na parte de baixo e a as atividades subsequentes na lateral.

Inicialmente são definidos os objetivos que o protótipo visa atender, bem como o que não está no escopo do projeto. Na etapa de definição é relacionado o que de fato vai ser representado de funcionalidades através do projeto, esta etapa visa definir riscos, custos e cronogramas de entrega das etapas do projeto. Realizadas todas as definições o protótipo é de fato desenvolvido, baseado nas definições anteriores. E por fim a validação é realizada, a partir de um plano de avaliação, esta etapa é essencial para descoberta de novos requisitos, por exemplo (Sommerville, 2011).

Uma atenção que deve se ter com o uso de protótipos é que estes podem não ser exatamente uma versão prévia do sistema final, essas observações devem ser feitas antes de atividades como testes com *stakeholders*, deve-se deixar alinhados os objetivos relativos à construção do artefato, para que não se desvie do foco durante o processo (Wazlawick, 2013).

Em situações que o fornecimento de um conjunto de funcionalidades ao cliente (ou usuário), o modelo de **processo incremental** pode ser utilizado (Pressman, et al., 2016). Além da situação citada, o auto afirma que projetos que estão em fase de expansão, podem se utilizar deste modelo.

O modelo de processo incremental, trabalha com uma modelagem em que as atividades são dispostas linearmente (ou seja, uma atividade deve iniciar, após o término da anterior), com alguns marcos, definidos como entregas, em cada entrega é disponibilizada uma versão do sistema final ou um incremento do mesmo (Wazlawick, 2013).

A grande diferença é, que como em cada entrega apenas uma parte ou um conjunto das funcionalidades do sistemas vai ser disponibilizada, esse fluxo linear do modelo dura um tempo menor para ser realizado (Hirama, 2012). Esta forma de trabalho tem a vantagem de deixar claro para

o cliente por quais evoluções o sistema está passando, apresentando uma grande mudança em relação aos primeiros modelos, onde a visão de evolução do sistema, se apresentava bem distante. A Figura 5 apresenta uma visão de como as atividades dos processos são distribuídas em um projeto, em função das funcionalidades entregues.

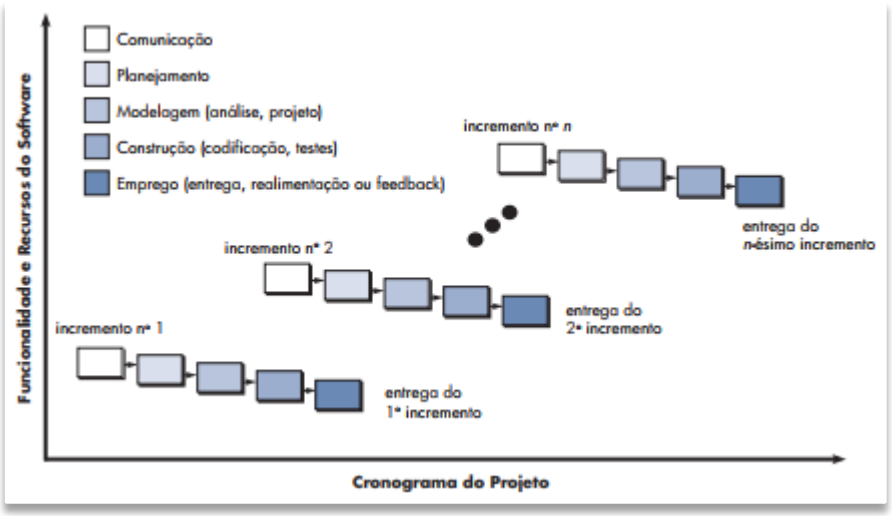

Figura 05 - Modelagem Incremental
Fonte: (Pressman, et al., 2016).
Descrição: A figura apresenta um gráfico com dois eixos, o eixo vertical representa as funcionalidades através dos incrementos, o eixo cresce em função do aumento do número de incrementos. O eixo horizontal, representa o tempo do projeto que aumenta em função do número de vezes em que as atividades do processo são realizadas. A figura também representa as atividades do modelo (comunicação, planejamento, modelagem, construção e emprego), em caixas com cores diferentes para cada atividade, cada incremento é representado pela sequencias das atividades descritas, representado um incremento do modelo.

A figura presenta uma visão do modelo, a partir de duas vertentes o crescimento do número de incrementos, representado o aumento das funcionalidades do sistema e a repetição das atividades de cada incremento. O modelo incremental, pode ser visto como uma variação do modelo cascata, pois mantém sua estrutura de atividades linear, ao mesmo tempo que pode ser visto também como evolução do modelo evolucionário, pois visa um planejamento focado nas entregas realizadas no ciclo (Wazlawick, 2013).

Sommerville (2011) apresenta as principais vantagens de se utilizar o modelo incremental:

- Os incrementos iniciais podem ser utilizados como protótipo, para melhorar os requisitos das versões posteriores.

- As vantagens com a utilização do sistema, já podem ser extraídas, a partir da primeira entrega, que possui as funcionalidades mais básicas.
- A forma de desenvolvimento incremental facilita a integração de novos módulos do sistema.
- Elementos mais importantes do sistema (desenvolvidos nos primeiros incrementos) passam pela etapa de testes mais vezes, pois sempre serão testados novamente antes da entrega ao cliente.

Outro modelo que se encaixa no contexto apresentado, é modelo espiral, nesse modelo de processo de software evolucionário são reunidas as naturezas iterativas da prototipação aos aspectos sistemáticos e controlados do modelo cascata.

Para saber mais sobre o modelo espiral, acesse o link ...
http://metodologiasclassicas.blogspot.com/p/blog-page.html

1.4.3 Modelo Unificado

O modelo de processo unificado surgiu para agregar de forma prática em um modelo de desenvolvimento de software, as melhores práticas dos modelos existentes. Para seus idealizadores a forma sequencial, que privilegia a arquitetura, a participação do cliente e entrega software de forma iterativa e incremental deve ser o ponto de partida essencial para o desenvolvimento de software moderno (Pressman, et al., 2016). O modelo unificado é o ponto de partida para os princípios utilizados nos modelos ágeis que compõem a próxima era dos modelos de desenvolvimento de software.

Segundo Sommerville (2011), o modelo unificado pode ser descrito através de três perspectivas, são elas:

- **Dinâmica**: mostrando as fases do modelo ao longo do tempo.
- **Estática**: mostrando as atividades realizadas no processo.
- **Prática**: sugerindo boas práticas que devem ser utilizadas no processo.

Wazlawick (2013) adiciona outras características importantes ao modelo unificado, que podem ser adaptadas para necessidades específicas de projetos ou empresas que utilizme o modelo, são elas:

- **Dirigido por casos de uso:** um caso de uso deve apresentar todos os aspectos de uma funcionalidade do sistema. Serve tanto para a guiar a equipe de desenvolvimento, com para o entendimento das funcionalidades do sistema por parte do usuário. Além das utilidades já citadas casos de uso, também podem servir para:
    - Definir e validar arquitetura do sistema
    - Auxiliar na criação de caso de testes
    - Planejar as próximas iterações do sistema
    - Servir de base para documentação do usuário
- **Centrado na arquitetura:** o modelo unificado prega que é preciso ter uma boa arquitetura, esta pode ser desenvolvida, através dos resultados da etapa de casos de usos. A arquitetura pode ser definida por um conjunto de classes traduzidas em forma de componentes que possivelmente tem uma relação. A arquitetura tem o objetivo de as estruturas das informações, operações, componentes e possivelmente as camadas do sistema proposto.
- **Iterativo e Incremental:** uma das praticadas herdadas dos modelos anteriores é a definição de incrementos para as entregas do sistema, no modelo unificado estes ciclos de entrega tem um tempo de duração definido e que em cada entrega uma parte significativa do sistema seja disponibilizada. Além de acrescentar um incremento ao software, esta etapa auxilia ampliando o conhecimento sobre os sistemas de forma geral, pois todas as fases (etapas) do desenvolvimento são executadas em cada novo incremento.
- **Focado em riscos:** no modelo unificado os casos de uso mais críticos são priorizados nos ciclos iniciais, afirmamos que o modelo é focado em riscos. Se um requisito gerado através de um caso de uso, apresenta um risco maior de desenvolvimento ou ao sistema, deve ser implementado primeiro, pois assim será possível tratar de forma antecipada problemas na integração com os demais módulos do sistema e também permite que passe por diversas vezes pela etapa de testes.

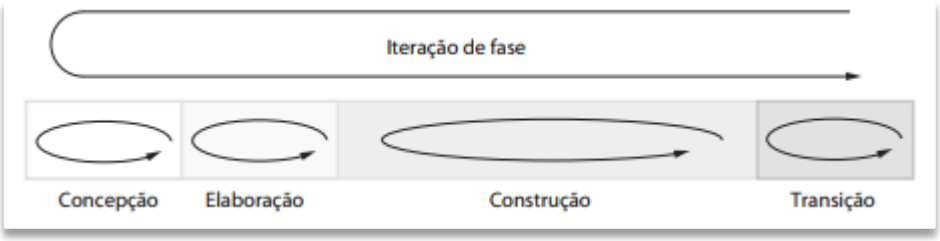

Figura 06 - Fases do Modelo Unificado
Fonte: (Sommerville, 2011).
Descrição: A figura apresenta quatro retângulos, representando cada fase do modelo: concepção, elaboração, construção e transição. Dentro de cada retângulo uma seta em direção circular, indicando que a fase é executada mais de uma vez como um ciclo. Acima da figura uma seta em direção circular com o texto iteração de fase, indica que as etapas representadas abaixo representam uma iteração do processo, que será executada diversas vezes.

As fases do modelo unificado são: concepção, elaboração, construção e transição, como representado na Figura 6. Estas etapas são executadas em cada ciclo, para a partir da escolha de uma quantidade significativa de casos de uso, prover um módulo ou um conjunto de funcionalidades do sistema. Vamos definir as etapas e atividades realizadas em cada uma.

Concepção

É a fase inicial do processo, nela devem ser identificadas todos os elementos e variáveis que farão parte do sistema. Essas informações auxiliarão nos levantamentos de requisitos iniciais e na construção de uma modelagem de solução inicial, através de um plano de desenvolvimento, baseados nos esforços e na estimativa de ciclos do projeto (Sommerville, 2011).

Um fluxo de trabalho para esta etapa, pode seguir as seguintes atividades:

- Coletar casos de uso para compor requisitos funcionais.
- Realizar análise de cenários, para auxiliar na coleta dos casos de uso.
- Separar casos de uso por prioridade para o negócio e complexidade de implementação.
- Após a separação, validar com o cliente quais casos dentre os mais complexos devem ser implementados inicialmente, partindo do ponto de vista do impacto para o negócio.

Fonte: (Wazlawick, 2013).

Elaboração

Após as etapas realizadas na fase inicial, as atividades da fase de elaboração são estabelecer a compreensão do problema, para em seguida modelar a arquitetura do sistema, realizar o planejamento do projeto, essas atividades auxiliarão a identificar os riscos do projeto. A etapa de elaboração deve produzir alguns artefatos, como modelo de requisitos, diagramas de casos de uso, modelos de arquitetura e um plano de desenvolvimento do sistema (Sommerville, 2011).

Uma lista das atividades em sequência das atividades desta etapa pode ser vista a seguir:

- Produção da arquitetura do sistema.
- Criar modelo de requisitos com um conjunto inicial de casos de uso.
- Criação de um projeto para a fase seguinte (construção).
- Selecionar as ferramentas utilizadas nas próximas etapas.
- Criar um plano de gerenciamento de riscos.

Construção

Nesta fase o software deve ser de fato desenvolvido, baseado no resultado das etapas anteriores. O desenvolvimento de dá através de iterações que de forma incremental adicionaram os componentes planejados para o software (Wazlawick, 2013). O resultado da fase de construção é o software funcionando com toda a sua documentação, validos e em uso pelos clientes e usuários (Sommerville, 2011).

A construção é realizada a partir das definições geradas pelo modelo de arquitetura, que vão guiar as modelagens dos componentes (sejam novos ou atualizações de componentes existentes), a ideia é que cada componente represente um caso de uso, que foi gerado nas etapas anteriores, a escolha dos caso de uso e componentes que serão implementados definem um ciclo de desenvolvimento ou uma iteração. Em seguida o código fonte é gerado, para que os recurso componentes definidos para o incremento possam ser realizados, após a fase de implementação dos componentes, os testes de unidade planejados são executados. Ao final dos testes de unidade os componentes são integrados ao sistema existente e os testes de integração são executados (Pressman, et al., 2016).

Transição

Esta fase inicia nas últimas etapas da fase de construção durantes as atividades relacionadas a integração do sistema e seus testes. Através de entregas, denominadas de *releases*, versões do sistema, juntamente com uma documentação atualizada, são liberadas ao cliente. As primeiras versões destes *releases* são denominadas versão beta. Inicialmente poucos usuários tem acesso as versões *beta* de um sistema, a medida em que estão mais testados, tornando o sistema mais robusto, são liberados para um público maior (Hirama, 2012).

A transição prevê também a evolução dos sistemas, que trata das evoluções dos sistemas após entrar em operação em um ambiente de produção. As atualizações tratam de correções de erros que não puderam ser corrigidos nas fases de implementação, bem como a implementação de novas funcionalidades (Wazlawick, 2013).

Uma implementação detalhada do modelo unificado pode ser encontrada no *Rational Unified Process* (RUP), por ser tão fiel ao modelo unificado, são tratados em alguns casos como sinônimos.

Para entender melhor o RUP, acesse o link a seguir
https://www.devmedia.com.br/rup-rational-unified-process/4574

1.4.4 Modelos Ágeis

No inicio dos anos 2000, alguns engenheiros de software, observaram que os modelos de processos existente, já não atendiam algumas necessidades relacionadas ao desenvolvimento de software atual e resolveram criar um movimento batizado de aliança dos ágeis, o grupo estaria fundamentado nos seguintes valores:

- As interações e os indivíduos estariam acima de processos e ferramentas.
- Software executando estaria acima de uma documentação completa.
- A colaboração dos clientes estaria acima de uma negociação contratual.
- Respostas a mudanças estaria acima de seguir um planejamento rígido.

Fonte: (Pressman, et al., 2016).

Os modelos de processo de desenvolvimento de software ágil, diferente dos modelos vistos até agora, não estão relacionados com descrição das atividades que serão executadas do inicio até a entrega do software ao cliente, bem como sua atualizações, estão relacionados ao fatores humanos que podem auxiliar ao desenvolvimento de um sistema com foco no resultado que deve ser alcançado (Wazlawick, 2013). Não significa que processos, ferramentas, documentação, negociação e planejamento não são importantes em um processo de desenvolvimento de software, apenas que estes novos elementos deveriam ser mais valorizados.

O resultado dessa aliança foi um documento com 12 itens, denominado manifesto ágil, que apresentava quais deveriam ser os princípios utilizados para o desenvolvimento de software, pois não adiantaria ter um processo de software bem estruturado se não fosse seguido à risca ou software com uma documentação completa se não funcionasse corretamente.

Wazlawick (2013), apresenta os 12 itens do manifesto ágil:

1. A maior prioridade deve ser satisfazer o cliente com entrega contínua e rápida de um software que agrega valor ao usuário.
2. As mudanças de requisitos devem ser bem vistas, como um diferencial competitivo para o cliente.
3. As entregas de software devem ser de forma frequente, em intervalos que variam de duas semanas a dois meses.
4. Analistas de negócio e equipes de desenvolvimento devem trabalhar em conjunto de forma diária.
5. Crie um bom ambiente com o suporte devido, isto motivará a equipe a desenvolver um bom trabalho.
6. A melhor forma de comunicar-se com a equipe é com uma interação pessoal.
7. A melhor medida de progresso de um projeto é o software funcionando.
8. Projetos ágeis devem ter desenvolvimento sustentado, seu ritmo deve se manter de forma constante pelos seus participantes.
9. A agilidade é melhor desenvolvida com uma boa atividade técnica e um bom *design*.
10. Diminuir ao máximo trabalhos não realizados.

11. Equipes autogerenciáveis geram melhores requisitos, arquiteturas e projetos.
12. As equipes devem de forma regular, refletir sobre melhorias e ajustes na forma de trabalho para atingir as metas.

Todo esse contexto apresentado, gerou uma série de novos modelos de desenvolvimento de sistemas, cada um aplicando de forma particular os itens apresentados, adaptando-os às suas necessidades.

Scrum

O Scrum é um modelo de desenvolvimento ágil de software dos mais utilizados, baseado nos princípios do manifesto ágil, consiste nas seguintes atividades: requisitos, análise, projeto, evolução e entrega. Em cada etapa citada, algumas atividades são realizadas e um ciclo passando por todas as etapas é denominado *sprint* (Pressman, et al., 2016). Em cada *sprint* uma parte da solução deve ser entregue ao cliente e adicionada na solução existente. Um projeto que utiliza Scrum, utiliza quantos *sprints* forem necessárias para finalizar o produto, o número de sprints varia com o tamanho do projeto.

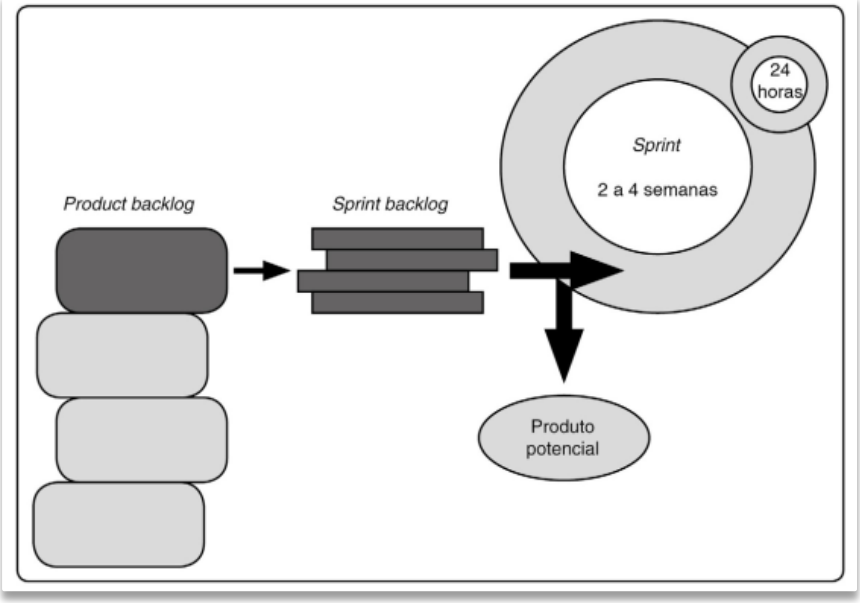

Figura 07 - Modelagem do Scrum
Fonte: (Wazlawick, 2013).
Descrição: A figura apresenta a modelagem do processo Scrum, do lado esquerdo da figura, quatro figuras retangulares, representam o backlog do produto (ou necessidades que o projeto visa atender), uma seta liga o backlog do produto ao

sprint backlog, no cento da figura, que representa as necessidades utilizadas na sprint atual (representadas por quanto retângulos sobrepostos, o que indica que uma lista de atividades). Por fim do lado direito um círculo cinza com um círculo interno branco escrito sprint 2 a 4 semanas, representa um tempo médio de duração da sprint, sobreposto ao círculo cinza maior, no mesmo modelo porém menor um círculo cinza com um círculo branco interno escrito 24 horas, representa o tempo de intervalo para a realização de uma reunião de verificação. Uma seta partindo do círculo maior aponta no sentido de cima para baixo para uma figura oval coma escrita produto potencial, representa a entrega ao final da sprint.

Um projeto no modelo Scrum, funciona com apresentado na Figura 7 anterior, um *backlog* do produto, com as necessidades expostas pelo cliente. Na reunião de planejamento, esse *backlog* é utilizado para gerar *o sprint backlog*, ou seja, as funcionalidades que serão implementadas na *sprint* atual e quanto tempo será necessário para desenvolvê-la. Após essas definições a sprint tem inicio de fato. A cada 24 h uma reunião de acompanhamento é realizada, para verificar o andamento de cada membro da equipe e possíveis impedimento que precisem ser tratados para que o andamento do projeto não seja interrompido. Ao final da sprint (entrega o que foi acordado com o cliente), uma reunião de revisão é feita, com intuito de reavaliar o trabalho realizado e possíveis melhorias no andamento do projeto. Para o inicio de uma nova sprint, uma nova reunião de planejamento é realizada e o processo recomeça, até que se atinja uma versão final, do produto e o projeto se encerre (Pressman, et al., 2016).

Para a execução do projeto o Scrum prevê alguns papéis a serem desempenhado são eles:

- ***Scrum Master*:** funciona como um facilitador, para gerenciar os conflitos existentes, deve conhecer bem os processos e a forma de trabalhar do Scrum.
- ***Product Owner***: conhece as necessidades do cliente e gerencia os requisitos que serão implementados em cada *sprint*.
- ***Scrum Team:*** equipe de desenvolvimento, que interagem para o desenvolvimento do produto de forma conjunta, não deve ultrapassar 10 membros por equipe.

O Scrum utiliza padrões de desenvolvimento de software, que se tornaram eficazes em projetos com prazos de entrega curtos e negócios mais críticos com requisitos em constante mudança (Pressman, et al., 2016).

Como já citado, existem diversos processos de desenvolvimento, baseado nos princípios dos modelos ágeis, para entender e conhecer mais alguns, acesse o link a seguir.
https://www.iebschool.com/pt-br/blog/software-de-gestao/as-metodologias-ageis-mais-utilizadas-e-suas-vantagens-dentro-da-empresa/

Nesta competência entendemos um pouco mais do que se trata a engenharia de software, bem como seus modelos são utilizados para atingir estes objetivos. Na próxima competência, veremos aspectos relacionados a garantia de qualidade do produto que é desenvolvido e entregue pelas equipes.

Bons estudos!

2.Competência 02 | Projeto de Fluxo de Dados e Qualidade de Software

Nesta semana vamos entender um pouco mais sobre qualidade de software em dois aspectos: a qualidade do produto que é entregue, e a qualidade do processo utilizado pelas equipes de desenvolvimento. Quais os aspectos mais relevantes e os modelos de avalição utilizados nos dois casos. Realize a leitura da competência, assista a vídeo aula, acesse o material complementar indicado, pois ele também é importante para a construção do conhecimento no nosso modelo de aprendizado, realize as atividades propostas e em caso de dúvida nos procure nos fóruns.

2.1 Projeto de Fluxo de Dados

É uma forma visual para descrever a relações de armazenamento de dados dentro de um sistema. Utiliza símbolos para representar a entrada e a saída dos dados em uma aplicação, servem para que se possa ter uma visualização prévia de ações que o sistema vai desempenhar, auxiliando engenheiros de sistemas nas atividades de desenvolvimento (Lucidchart, 2019).

A técnica de utilização de diagramas para representar fluxo de dados surgiu no final da década de 70. Impulsionado por dois conceitos utilizados na área de desenvolvimento de sistemas: a análise orientada a objetos e a modelagem de sistemas estruturados, ambas são técnicas utilizadas para auxiliar a construção de sistemas (Macoretti, 2019).

Outro elemento relacionado à utilização de Diagramas de Fluxo de Dados (DFD) é o da *Unified Modeling Language* (UML), a UML é uma linguagem utilizada para representar visões detalhadas de um sistema orientado a objetos, foi um legado deixado pelo Modelo Unificado, apresentado na competência 01 da disciplina. É importante não confundir os conceitos o DFD foca exclusivamente no fluxo dos dados dentro de um sistema, a UML apresenta visões em diversas perspectivas como classes, objetos, atividades, componentes etc. Por muitas vezes as técnicas podem ser combinadas como forma de ampliar os recursos relacionadas a modelagem do sistema (DiarioUML, 2019).

Para entender melhor como funciona um DFD, vamos utilizar o modelo de Yourdon e Coad, pois é um modelo de referência e amplamente utilizado para criar DFDs. O modelo apresenta quatro elementos: entidade externa, processo, armazenamento de dados e fluxo de dados. A seguir

vamos detalhar cada um dos elementos, apresentados na Figura 8, e apresentar exemplos de utilização.

Entidade Externa: qualquer elemento externo ao sistema (pessoa ou outro sistema), que envie ou receba dados do sistema. Pode ser entendido como o destino dos dados que entram e saem do sistema, é representado por um quadrado nas extremidades do sistema.

Processo: procedimento que realiza alteração nos dados, pode ser um cálculo, classificação ou qualquer outra atividade que precise manipular dados presentes no sistema.

Armazenamento de Dados: elementos que realizam o armazenamento dos dados, geralmente é utilizado um banco de dados para realizar esta atividade.

Fluxo de Dados: pode ser definido como a rota que os dados realizam entre as entidades anteriores: entidades externas, processos e o armazenamento. É representado por uma seta na modelagem.

Fonte: (Lucidchart, 2019).

Para a criação de um DFD, utilizando o modelo descrito, devemos seguir algumas orientações:

- Cada processo dever ter, pelo menos, uma entrada e uma saída.
- Cada armazenamento de dados deve ter, pelo menos, um fluxo de entrada e um de saída.
- Dados armazenados em um sistema devem passar por um processo.
- Todos os processos em um DFD vão para outro processo ou um armazenamento de dados.
- Dados armazenados em um sistema devem passar por um processo.

Fonte: (Lucidchart, 2019).

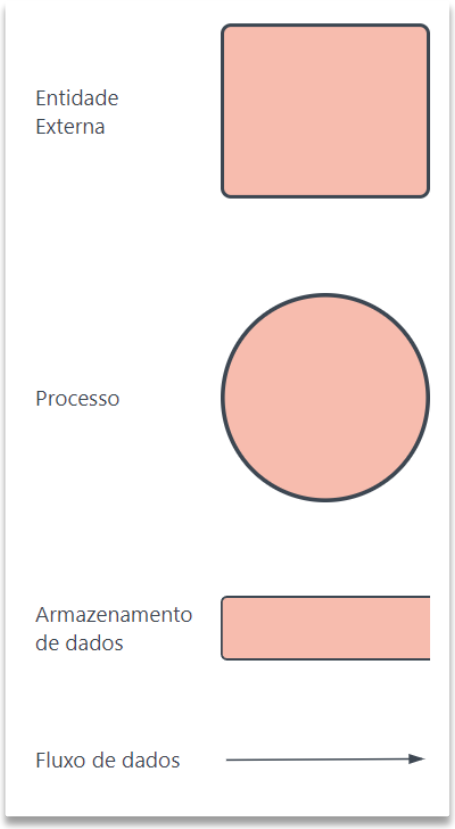

Figura 08 - Elementos de um DFD
Fonte: Adaptado de LUCIDCHART.
Descrição: a figura apresenta os elementos presentes em diagrama de fluxo de dados, do lado esquerdo os nomes das entidades pela ordem de cima para baixo: entidade externa, processo, armazenamento de dados e fluxo de dados. Do lado direito pela ordem de cima para baixo temos: quadrado representando a entidade externa, círculo representado o processo, retângulo representado o armazenamento de dados e uma seta representando o fluxo de dados.

A Figura 9 a seguir, representa uma modelagem genérica de um DFD, na modelagem são apresentados como as entidades externas, processos e repositório de dados são modelados, para a representação de um DFD.

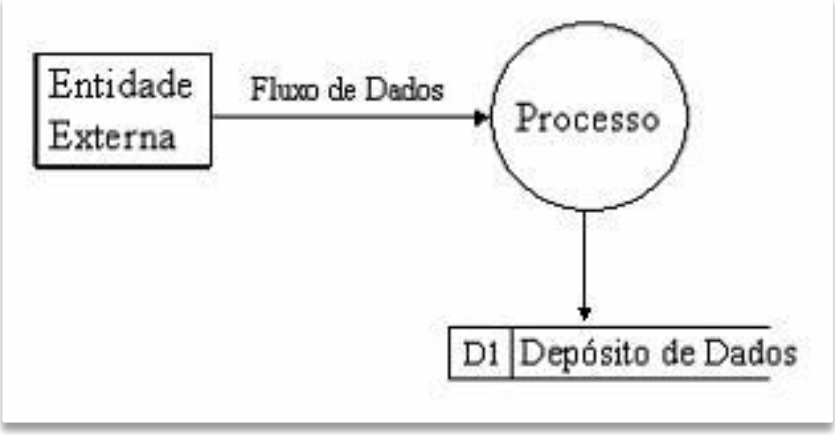

Figura 09 - Representação genérica de um DFD
Fonte: adaptado de (Macoretti, 2019).
Descrição: A figura representa uma modelagem genérica de um diagrama de fluxo de dados, apresentado uma entrada externa, ligada a um processo, este processos levam a um repositório de dados, as ligações entre o processo e entidade externa, bem como as ligações entre processo e repositório de dados, são feitas por fluxo de dados, representados por setas.

Esta representação é bem genérica, apenas para exemplificar como os elementos podem se relacionar. As modelagens podem ser mais elaboradas, baseados nos níveis de abstração que utilizam, vamos entender como podem ser classificadas:

- **Nível 0:** representa uma visão básica do sistema modelado. A modelagem deve ser de alto nível, mostrando visões dos sistemas para analistas de negócios, dados e equipes de desenvolvimento.

- **Nível 1:** deve oferecer mais detalhes que o nível anterior, deve destacara as funções mais importantes do sistema, neste nível é comum a presença de subprocessos, ou seja, a divisão de processos maiores em partes.

- **Nível 2:** este nível se diferencia dos demais apenas por aumentar a profundidade da descrição, não existe novos elementos, apenas fica maior em virtude da necessidade de acrescentar mais detalhes a modelagem.
  Fonte: (Lucidchart, 2019).

O exemplo a seguir, Figura 10, apresentas um exemplo de uso real modelado através dos conceitos de um DFD. É possível identificar, quais as etapas do processo registrar pedido, quais os elementos envolvidos e o fluxo dos dados para que a operação seja realizada com sucesso.

O DFD apresentado pode ser entendido como um a regra de negócio de um sistema de compras. A regra funciona da seguinte maneira: o cliente realiza um pedido o processo recebe os dados do cliente do repositório dados_cliente e registra os dados do pedido no repositório dados_pedido. Para finalizar o processo envia um comprovante com as informações coletadas para o elemento externo cliente.

Figura 10 - Exemplo de um DFD
Fonte: (DiarioUML, 2019).
Descrição: a imagem representa um DFD para registro de um pedido de um cliente, que é o elemento externo. A modelagem apresenta um processo chamado registrar pedido, representado por um círculo no centro da figura, no lado esquerdo o elemento esterno cliente é relacionado com o processo registrar pedido, pelos fluxos de entrada chamado de pedido e de saída chamado comprovante. No lado direito os repositórios de dados cliente e pedidos se relacionam com o processo registrar pedido, através dos fluxos de dados, dados do cliente e dados do pedido. A regra funciona da seguinte maneira: o cliente realiza um pedido o processo recebe os dados do cliente do repositório dados_cliente e registra os dados do pedido no repositório dados_pedido. Para finalizar o processo envia um comprovante com as informações coletadas para o elemento externo cliente.

O DFD apresentado na Figura 10, pose ser visto como de nível 0, pois apresenta de forma direta as etapas do processo de registro de um pedido de compra, da forma como está modelado, se apresentado a um desenvolvedor ou a um analista de negócios, ambos poderiam ter o entendimento do que se deseja com a operação descrita.

A utilização de um DFD, pode ser vista de diversas formas com objetivos diferentes. Na engenharia de software, pode ser utilizada para auxiliar o desenvolvimento técnico, como uma modelagem inicial do sistema. Outras formas de utilização de um DFD são:

- Analisar sistemas existentes para encontrar melhorias.
- Analisar um fluxo de dados de forma que este se torne eficiente, baseado e uma regra de negócio.
- Para demonstrar requisitos técnicos e de negócios, auxiliando na composição da solução a ser implementada.
- Analisar estruturas de um sistema, seja com o objetivo de entendê-lo melhor ou para propor melhorias técnicas ou de regras de negócio.

O entendimento inicial que precisamos ter é que modelar fluxos de dados, pode auxiliar nos resultados coletados nas etapas posteriores. A qualidade de um sistema, está relacionada à dois itens como veremos mais a frente, ao que o sistema entrega a ao processo utilizado para desenvolvê-lo. Quando deixamos de modelar corremos o risco de criar elementos desnecessários e construir entendimentos essenciais para a produção de um software.

2.2 Qualidade de Software

O objetivo da engenharia de software é produzir um produto com qualidade em um tempo razoável e com um custo adequado, para atingir tal objetivo ao longo tempo com a evolução dos modelos de desenvolvimento de software, foram criadas métricas para avaliar a qualidade não apenas do produto, mas também das metodologias utilizadas, nos próximos tópicos vamos entender como funciona cada uma dessas áreas.

2.2.1 Qualidade do Produto

Problemas com o software sempre existiram, pois com o início de sua produção ele era lento, pouco confiável, difícil de usar, manter e reutilizar. Com o intuito de resolver estes problemas surgiu como visto anteriormente a engenharia de software com o objetivo de se encontrar uma maneira sistemática de se produzir software com qualidade (Sommerville, 2011).

Para entender mais sobre a qualidade do produto vamos utilizar a norma ISO 25010:2011, a norma define um conjunto de oito características relacionadas a elementos internos e externos, mais cinco características relacionadas ao software em uso, vamos entender melhor cada uma delas e suas subdivisões.

Características do Produto

São as características internas e externas relacionadas ao ambiente de desenvolvimento, a norma as distribui da seguinte forma:

1. **Adequação** funcional**:** mede grau de funções disponibilizadas por um produto para atender necessidades sob condições específicas. Pode ser subdividida em três partes.
    a. **Completude funcional:** o software deve apresentar todas as funcionalidades necessárias para atingir seus objetivos.
    b. **Corretude funcional:** O software deve gerar dados e consultas de forma correta de acordo com as suas definições.
    c. **Funcionalidade apropriada:** analisa se as funcionalidades do sistema facilitam a realização das tarefas para quais foram especificadas.

2. **Confiabilidade:** um software confiável, mantem um comportamento de acordo com o esperado ao longo do tempo, esta característica está ligada também a redução de defeitos e o seu comportamento em situações fora do normal. Outros itens relacionados a esta característica são:
    a. **Maturidade:** um sistema maduro com o passar do tempo, apresenta menos defeitos, sua maturidade deve aumentar com o tempo. Para tal é preciso um bom processo de manutenção e a realização dos testes de regressão.
    b. **Disponibilidade:** o software deve sempre estar disponível para uso, quando for necessário.
    c. **Tolerância a falhas:** avalia o comportamento do software em situações fora do normal, como uma falha por exemplo. Falha não está relacionada a um defeito do sistema, mas sim a uma situação externa e inevitável. Um software tolerante a falhas deve ter planos de ação nessas situações.
    d. **Recuperabilidade:** esta característica está relacionada com a recuperação de dados e o pleno funcionamento em casos de desastre. Esta característica não está relacionada com falhas temporárias, mas casos em que os danos possam não ser reversíveis.

3. **Usabilidade**: avalia em um sistema a existência de elementos que permitam ao usuário ter o entendimento e utilizar o sistema, bem como ser atrativo. Estes itens são avaliados através das seguintes características:
    a. **Apropriação reconhecível**: característica que permite ao usuário identificar se o sistema é apropriado a sua necessidade.

b. **Inteligibilidade**: característica relacionada ao grau de facilidade que o usuário encontra para se apropriar aos principais conceitos do sistema.
c. **Operabilidade**: característica que avalia o quão fácil é utilizar o sistema.
d. **Proteção contra erro do usuário**: característica que avalia se o sistema foi projetado para evitar que o usuário cometa erros.
e. **Estética de interface:** característica que avalia se a interface proporciona uma boa interação com o usuário.
f. **Acessibilidade:** característica que avalia se o sistema atende necessidades de usuários especiais.

4. **Eficiência de Desempenho:** a característica da eficiência avalia o quão bem os recursos de tempo e espaço são utilizados em um sistema, avalia se os recursos são utilizados de forma otimizada. Esta característica é avaliada, através dos itens:
   a. **Comportamento em relação ao tempo:** avalia o tempo que o sistema leva para processar suas funções.
   b. **Utilização de recursos:** ao contrário do primeiro item, avalia a eficiência de recursos em relação ao armazenamento ou utilização de memória. Pode incluir também outros recursos necessários ao funcionamento do sistema como banda de transmissão de rede.
   c. **Capacidade:** esta característica avalia o desempenho dos sistemas dentro dos limites estabelecidos nos seus requisitos. Por exemplo, se dispôs a realizar 20 transações de forma simultânea, como se comportaria de fossem realizadas mais de 20 transações.

5. **Segurança:** esta característica avalia se as funcionalidades e os dados presentes no sistema são protegidos e com acesso, apenas a pessoas autorizadas. Não confundir com a qualidade de uso seguro, relacionadas a questões com questões de segurança pessoal, instalações e meio ambiente. A avaliação do critério é baseada nos cinco itens listados a seguir:
   a. **Confidencialidade:** apenas pessoas autorizadas devem ter acessos aos dados e funcionalidades do sistema.
   b. **Integridade:** os dados e funcionalidades devem ser protegidas contra pessoas não autorizadas.
   c. **Não repúdio:** permite constatar que determinada ação ou acesso foi realizada de fato, esse fato não pode ser desfeito ou alterado posteriormente.
   d. Rastreabilidade de uso: esta característica avalia as formas que um sistema rastreia as ações realizadas, seja por um usuário ou por outro sistema.
   e. **Autenticidade:** em um sistema é característica que verifica se uma pessoa que realiza um acesso é de fato quem diz ser, o mesmo se aplica por exemplo aos dados, estes também precisam ser validados antes que o sistema realize algum tipo de processamento.

6. **Compatibilidade**: avalia se dois ou mais sistemas ou componentes de um sistema, interagem entre si, em um ambiente compartilhado de hardware/software. Esta avaliação é baseada em dois critérios:

a. **Coexistência**: avalia se um sistema realiza suas atividades, enquanto compartilha recursos com outros (sistemas ou componentes), sem impactos negativos.
b. **Interoperabilidade**: avalia a qualidade da comunicação de um software com outro que se espera que haja interação. Se essa interação for de baixa qualidade, afirmamos que ele não tem interoperabilidade.

7. **Capacidade de Manutenção:** avalia a facilidade de se realizar manutenção no sistema, seja correção de um erro ou evolução do mesmo. Esta característica é percebida pelas equipes de desenvolvimento, mas pode afetar a sua utilização por parte dos clientes. A avaliação é baseada nas seguintes características:
   a. **Modularidade:** avalia a subdivisão do sistema em partes menores, as mudanças em uma dessas partes devem ter o menor grau de impacto nas demais partes do sistema.
   b. **Reusabilidade:** avalia quais partes do sistema podem ser utilizadas para construir outros sistemas.
   c. **Analisabilidade:** característica que avalia o quão fácil é depurar (identificar erros ou falhas) um sistema.
   d. **Modificabilidade:** capacidade que o sistema oferece para correção de erros ou alterações, sem essas ações gerem novos defeitos ou afetem sua estrutura interna existente.
   e. **Testabilidade:** característica relativa ao processo de desenvolvimento, pois verifica a facilidade de se realizar testes de regressão em um sistema.

8. **Portabilidade:** avalia em que grau o software pode ser de fato transferido de um ambiente para outro. Suas características são:
   a. **Adaptabilidade:** o software é integrado ao novo ambiente sem que outras ações precisem ser realizadas.
   b. **Instalabilidade:** verifica a facilidade no processo de instalação do software.
   c. **Substituibilidade:** esta característica verifica se se um sistema pode substituir outro sistema, no mesmo ambiente e realizando as mesmas operações.

   Fonte: (Wazlawick, 2013).

Características do Uso

São características relacionadas ao software quando estiver em uso no seu ambiente final. Pois seria difícil avaliar tais itens no ambiente de produção (Wazlawick, 2013). As características são divididas em 5 categorias, vamos entender como funcionam essas categorias e suas subdivisões.

1. **Efetividade:** esta característica avalia se o sistema auxilia o usuário a atingir seus objetivos de forma correta.
2. **Eficiência:** a eficiência avalia a relação entre o investimento feito pelo cliente e o retorno proporcionado pelo sistema, essa medida pode ser financeira ou utilizar outros parâmetros de avaliação.

3. **Satisfação:** a característica avalia o grau de satisfação do usuário em relação ao uso do produto. A avaliação considera os seguintes itens:
    a. **Utilidade:** avalia o grau de satisfação com os resultados decorrentes do uso do sistema.
    b. **Prazer:** avalia o grau de prazer na utilização do sistema para resolução de um problema.
    c. **Conforto**: a característica se refere ao conforto mental e físico na utilização do sistema.
    d. **Confiança:** avalia o quanto os usuários confiam no sistema e no que é proposto por ele.
4. **Uso sem riscos:** verifica se o sistema atende os níveis de segurança relativos a pessoas, negócios e meio ambiente. Subdivide-se em:
    a. **Mitigação de risco econômico:** avalia o grau de riscos financeiros relativos à propriedade e reputação de seus usuários.
    b. **Mitigação de riscos à saúde:** avalia o grau de riscos relativos à integridade física das pessoas relacionadas ao uso do sistema.
    c. **Mitigação de risco ambiental:** avalia o grau de riscos relativos à riscos ambientais e a propriedade.
5. **Cobertura de contexto:** avalia o grau de uso do sistema de forma efetiva, eficiente, sem riscos e com satisfação nos contextos do sistema, previamente descritos, e em outros contextos. A avalição possui duas características:
    a. **Completude de contexto:** avalia o grau de uso do sistema de forma efetiva, eficiente, sem riscos e com satisfação nos contextos de utilização especificados pelo sistema.
    b. **Flexibilidade:** avalia o grau de uso do sistema de forma efetiva, eficiente, sem riscos e com satisfação nos contextos diferentes de utilização previstos pelo sistema.

Fonte: (Wazlawick, 2013).

2.2.2 Qualidade do Processo

A qualidade de produtos de software pode ser fortemente afetada pela qualidade do processo usado para desenvolvê-los. Dentre os modelos de processo já vistos, como modelo unificado, métodos ágeis, incremental etc. Todos apresentam vantagens e desvantagens, e cada um deles pode ser mais bem aplicado em determinadas situações do que outros.

A melhoria do processo de software engloba um conjunto de atividades que levarão a um melhor processo de software e, em consequência, uma maior qualidade do software fornecido, dentro do prazo de entrega (Hirama, 2012).

Deve-se, porém, diferenciar a questão do modelo teórico em si da questão relacionada à implementação do modelo em uma empresa específica. Ou seja, um modelo pode ser adequado, mas a empresa pode estar utilizando de forma inadequada. Em função dessa observação, foram

definidos modelos de avaliação de qualidade da implementação de processos nas empresas (Sommerville, 2011).

A melhoria do processo de software engloba um conjunto de atividades que levarão a um melhor processo de software e, em consequência, uma maior qualidade do software fornecido, dentro do prazo de entrega. O resultado é um processo de software melhorado que leva a uma qualidade mais alta do software. O software que a empresa produz será fornecido com menos defeitos, o retrabalho em cada estágio do processo de software será reduzido e a entrega no prazo se tornará muito mais possível (Wazlawick, 2013).

Nesse contexto, um modelo de maturidade permite que uma empresa classifique a sua maturidade em relação a um processo específico da organização. Possibilitando que a empresa possa obter os seguintes objetivos: (1) medir o nível de maturidade atual; (2) quais níveis de maturidade pretende alcançar e (3) quais erros nos processos podem ser eliminados (Pressman, et al., 2016).

Esses modelos não prescrevem este ou aquele ciclo de vida, mas avaliam quão bem uma empresa está aplicando e gerenciando seu processo de desenvolvimento com o modelo de processo escolhido.

CMMI - Capability Maturity Model Integration

É uma abordagem para a melhoria de processos e foi construído de forma independente, com participação da indústria, do governo norte-americano e do Instituto de Engenharia de Software (SEI).

Os modelos CMMI podem ser usados como guias para desenvolver e melhorar processos da organização, e como um *framework* para avaliar a maturidade dos processos da organização (Sommerville, 2011).

O CMMI se originou na indústria de software, mas também tem sido adaptado a outras áreas, como a indústria de hardware, serviços e comércio em geral. O termo “software” sequer aparece nas definições de CMMI, o que torna o modelo bem mais abrangente que os seus antecessores.

Existem duas representações do CMMI, a representação contínua e a representação em estágios.

A representação contínua é projetada para permitir à empresa focar em processos específicos que deseja melhorar em função de suas prioridades. Já a representação em estágios é aplicada à organização como um todo e permite que se compare a maturidade de diferentes organizações.

A avaliação pela representação contínua mede a capacidade da empresa em relação a um ou mais processos. Já a avaliação em estágios mede a maturidade da empresa.

Nível	Capacidade	Maturidade
0	Incompleto	
1	Realizado	Inicial
2	Gerenciado	Gerenciado
3	Definido	Definido
4		Quantitativamente Gerenciado
5		Em Otimização

Tabela 01 - Níveis de Capacidade e Maturidade do CMMI
Fonte: (Wazlawick, 2013).
Descrição: A tabela apresenta os níveis de capacidade e maturidade do CMMI.

Níveis de Capacidade

Existem, portanto, quatro níveis de capacidade para processos no CMMI, como apresentado na Tabela 1. Um nível de capacidade é atingido quando os objetivos genéricos daquele nível são atingidos. São eles:

- **Nível 0 – Incompleto:** Pode ser tanto um processo que não foi estabelecido quanto um processo que não é executado de forma adequada. Um ou mais dos objetivos específicos da área de processo não são satisfeitos, e não existem objetivos genéricos, já que não existe razão para institucionalizar um processo parcialmente realizado.

- **Nível 1 – Realizado:** é um processo seguido, mas que ainda não foi institucionalizado. Por esse motivo, a empresa corre o risco de perder essa conquista, caso não avance para os níveis seguintes.

- **Nível 2 – Gerenciado**: é realizado de acordo com um planejamento e uma política definidos. Usa recursos humanos capacitados e produz produtos de forma previsível. Envolve os interessados relevantes, é monitorado, revisado e controlado. A aderência dos projetos ao processo é avaliada. Esse nível garante que as práticas sejam mantidas mesmo em períodos de estresse.

- **Nível 3 – Definido:** é gerado a partir de um conjunto de processos padrão da organização, de acordo com as regras de geração de processos definidas. Sua descrição é mantida e sua evolução pode contribuir para o patrimônio de processos da empresa.
  Fonte: (Wazlawick, 2013).

Níveis de Maturidade

- **Maturidade Nível 1 – Inicial:** neste nível a empresa não possui padronização em seus processos, tudo é realizado de forma *ad hoc*. O gerenciamento na empresa é realizado de forma pontual para os processos e não há organização, o que pode gerar um abandono dos processos em tempos de crise. O sucesso da empresa depende mais das capacidades individuais de seus funcionários do que de processos bem estabelecidos.

- **Maturidade Nível 2 – Gerenciado:** aqui foram desenvolvidos alguns processos, porém a mesma tarefa é executada por diferentes pessoas. Além disso, não há treinamento formal sobre a implantação dos processos padronizados, há uma grande dependência no conhecimento dos colaboradores e os erros são muito comuns nos processos.

- **Maturidade Nível 3 – Definido:** nesta fase há padronização e documentação dos processos e foram realizados treinamento para comunicar aos colaboradores o funcionamento dos processos e formalização das práticas existentes na empresa.

- **Maturidade Nível 4 – Quantitativamente Gerenciado:** há um monitoramento dos processos padronizados e realiza algumas ações onde os processos não estão funcionando de forma adequada. Aqui os processos estão em melhoria contínua e tem-se o fornecimento das melhores práticas. Algumas ferramentas auxiliam na automatização do processo.

- **Maturidade Nível 5 – Em Otimização:** Aqui é o máximo da maturidade da empresa. Houve a refinação dos processos até se chegar a boas práticas, com base nos resultados da melhoria contínua do nível anterior e nos modelos de maturidade empregados em outras empresas. A TI tem sua importância quando automatiza os fluxos de trabalhos através do uso de ferramentas para melhorar a efetividade e a rápida adaptação dos colaboradores ao processo automatizado.
  Fonte: (WAZLAWICK, 2013).

Para entender como funciona na prática acesse o link e assista o vídeo abaixo
https://www.youtube.com/watch?v=BifAppF11YE

MPS.BR – Modelo de Referência para Melhoria do Processo de Software

O MPS.BR um modelo de avaliação de empresas produtoras de software brasileiro criado através de uma parceria entre a SOFTEX, o governo federal e academia (pesquisadores em geral). O modelo brasileiro é independente, mas compatível com Normas como o CMMI (Wazlawick, 2013).

A principal justificativa para a criação desse modelo foram os altos custos dos processos de avaliação ou certificação internacionais, que se tornam proibitivos para pequenas e médias empresas. Pois apresenta um custo significativamente mais baixo, por ter consultores e avaliadores

residentes no Brasil e pelo fato de que apresenta 7 níveis de maturidade em vez de apenas 5, como o CMMI (Wazlawick, 2013).

A escala de progressão na melhoria de processos possui degraus mais suaves, especialmente nos níveis mais baixos, ou seja, é possível subir um nível com menos esforço do que seria necessário para subir um nível no CMMI.

Os níveis de maturidade do MPS.BR são os seguintes:

- **G – Parcialmente Gerenciado:** neste ponto inicial deve-se iniciar o gerenciamento de requisitos e de projetos.

- **F – Gerenciado:** introduz controles de medição, gerência de configuração, conceitos sobre aquisição e garantia da qualidade.

- **E – Parcialmente Definido**: considera processos como treinamento, adaptação de processos para gerência de projetos, além da preocupação com a melhoria e o controle do processo organizacional.

- **D – Largamente Definido:** envolve verificação, validação, além da liberação, instalação e integração de produtos, dentre outras atividades.

- **C – Definido:** aqui ocorre o gerenciamento de riscos.

- **B – Gerenciado Quantitativamente:** avalia-se o desempenho dos processos, além da gerência quantitativa deles.

- **A – Em Otimização:** há a preocupação com questões como inovação e análise de causas.

  Fonte: (Wazlawick, 2013).

Para saber mais e verificar as diferenças entre os modelos apresentados, acesse o link
https://www.youtube.com/watch?v=sQYbj6tGnhQ

Nesta segunda competência entendemos um pouco mais sobre os fluxos de dados e sobre qualidade nas visões do sistema e do processo que é utilizado para que o sistemas possa ser desenvolvido e entregue, para ampliar seus conhecimentos nesses tópicos, assista a vídeo aula e acesse o material recomendado pelos professores no ambiente virtual de aprendizagem.

Bons estudos !!

Conclusão

Durante as duas semanas do nosso curso, entendemos um pouco mais sobre o processo de desenvolvimento de software, seus aspectos e características mais relevantes. Este entendimento para quem deseja trabalhar com desenvolvimento de sistemas é muito importante, pois como pudemos observar um bom software que atenda às expectativas do cliente, desenvolvido no tempo planejado e com um custo viável, precisa ser bem planejado.

Os conteúdos apresentados mais do que habilidades para a construção do código, visam criar um profissional que desenvolve sistemas com um pensamento voltado para entender o processo em que se encontra e qual o seu papel, na construção de um software com qualidade, bem como na evolução dos processos, pois estes também precisam ser revistos de tempo em tempos para atender novas demandas decorrentes das atividades de desenvolvimento e dos clientes que também são parte desse processo.

Por fim, esperamos que o que foi construído com a colaboração dos professores, se mantenha para além das esferas do nosso curso e os auxiliem na carreira profissional de cada um.

Bons estudos e até a próxima!

Referências

DiarioUML. **DiarioUML. site da DiarioUML**, 2019. Disponivel em: <https://diariouml.wordpress.com/2014/04/03/o-que-e-um-diagrama-de-fluxo-de-dados/>. Acesso em: 03 Outubro 2019.

HIRAMA, K. **Engenharia de Software Qualidade e Produtividade com Tecnologia**. Rio de Janeiro: Elsevier, 2012.

Lucidchart. Lucidchart. **Site da Lucidchart**, 2019. Disponivel em: <https://www.lucidchart.com/pages/pt/o-que-e-um-diagrama-de-fluxo-de-dados>. Acesso em: 03 Outubro 2019.

MACORETTI, J. C. Macoratti.net. **Macoratti.net**, 2019. Disponivel em: <http://www.macoratti.net/vb_dfd1.htm>. Acesso em: 03 Outubro 2019.

PRESSMAN, R.; MAXIN, B. **Engenharia de Software**. 7º. ed. São Paulo: McGraw Hill Brasil, 2016.

SOMMERVILLE, I. **Engenharia de Software**. 9°. ed. São Paulo: Pearson, 2011.

WAZLAWICK, R. S. **Engenharia de Software Conceitos e Práticas**. 1°. ed. Rio de Janeiro: Elsevier, 2013.

Minicurrículo do Professor

Pedro Henrique Barboza da Silva

Mestre em Informática Aplicada (2017) e Licenciado em Computação (2015) pela Universidade Federal Rural de Pernambuco, onde atua como pesquisador do grupo TECNES - Informação Educação e Tecnologia Colaborativa em Saúde. Atualmente é professor do curso técnico de Desenvolvimento de Sistemas, na modalidade EAD, na Secretaria de Educação do estado de Pernambuco e professor do ensino superior nos cursos de Sistemas de Informação e Análise e Desenvolvimento de Sistemas.

Aline Chagas Rodrigues Marques

Mestrado em Ciência da Computação no Centro de Informática (CIN) na Universidade Federal de Pernambuco (UFPE). Possui especialização em Análise de Sistemas pela Universidade Federal do Pará (2011) e graduação em Bacharelado em Ciência da Computação pelo Centro Universitário do Estado do Pará (2009). Atuou como Técnica em Informática no Instituto de Previdência e Assistência do Município de Belém (IPAMB). Realiza pesquisas com ênfase em Engenharia de Software, ITIL e atuou nos seguintes temas: ontologia, agentes de software e web semântica, ITIL e OpenUp/Basic.