FUNDAMENTOS DE DESENVOLVIMENTO DE SOFTWARES

AULA 1

Prof.ª Luciane Yanase Hirabara Kanashiro

SOFTWARE

CONVERSA INICIAL

Com certeza você já ouviu falar de software. Nós usamos algum software provavelmente todos os dias. Se você utiliza um computador, a Smart TV ou um celular, então você utiliza software.

Mas mesmo utilizando-o todos os dias, saberia definir o que ele é? O que diferencia software de sistema? Ou, ainda, o que o diferencia de um programa?

A seguir, veremos algumas definições que o auxiliará a responder às questões anteriores.

- No tópico 1, veremos sobre software, sistemas e programas.
- No tópico 2, veremos com mais detalhes os sistemas operacionais.
- No tópico 3, veremos sobre algoritmos e linguagens.
- No tópico 4, abordaremos sobre o jargão na área de TI.
- No tópico 5, ainda, veremos o ciclo de vida do desenvolvimento de software, onde serão abordadas diferentes metodologias que são frequentemente utilizadas.

TEMA 1 – SOFTWARE, PROGRAMAS E SISTEMAS

Com o advento da tecnologia e popularização da internet, os softwares, programas e sistemas têm sido uma constante em nossa vida. Provavelmente você utiliza algum tipo de software todo dia. Mas mesmo utilizando, você saberia definir o que ele é? Saberia diferenciar um software de um programa?

Nesse tópico, veremos diversas definições que o ajudará a entender as diferenças conceituais entre os termos software, programa e sistema.

1.1 SOFTWARE E PROGRAMAS

A seguir é dada uma definição técnica do que é um software. Observe que são vários conceitos que se unem na definição dada por Pressman (2001):

- Software consiste em:
    - instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados;

- estruturas de dados que possibilitam aos programas manipular informações adequadamente;
- informação descritiva, tanto na forma impressa quanto na virtual, descrevendo a operação e o uso dos programas.

Simplificando a definição anterior, podemos dizer que um software é um programa de computador e toda a documentação associada a esse programa.

Mas o que é um programa? Como você definiria um programa de computador?

Uma definição conhecida e frequentemente utilizada por todos da área de TI é de que é uma sequência de passos ou instruções definidas por um algoritmo.

Mas o que é um algoritmo? Aqui você pode assumir que este é como uma receita de bolo, um tutorial. Posteriormente, em outro tópico, veremos um conceito mais aprofundado.

Um software tem ainda uma classificação, na realidade, ele pode ser classificado ou categorizado de várias maneiras. Nesta etapa, utilizaremos uma adaptação da classificação dada por Marçula (2009):

- Software de sistema (básico):
  - sistemas operacionais;
  - software utilitário.
- Software aplicativo.

1.2 Software de sistema (básico)

Software básico consiste em uma coleção de programas para apoiar outros programas. Realiza as tarefas de gerenciamento necessárias ao seu funcionamento. Temos nessa categoria os sistemas operacionais e o software utilitário.

O **sistema operacional** é o software responsável pelo funcionamento geral dos sistemas de computação. Sem ele os sistemas de computação praticamente não funcionariam. Como exemplo de sistemas operacionais temos Windows, Unix, Linux e MacOS. A seguir são mostrados os ícones do Windows, Android, ubuntu e MacOS, respectivamente na ordem citada.

Devido à importância, falaremos mais adiante e com mais detalhes sobre o Sistema Operacional.

Os **softwares utilitários ou ferramentas de sistema** são os que acompanham o sistema operacional e o auxiliam nas suas tarefas (por exemplo, gerenciamento de dispositivos, mídias e programas).

O software utilitário fornece ao usuário ferramentas para organizar os discos, verificar a disponibilidade de memória e corrigir as falhas de processamento. Costumamos dizer que são softwares úteis ao sistema operacional, ou seja, que trabalham em função do sistema operacional.

São exemplos de utilitários: antivírus, compactadores, emuladores, desfragmentadores, formatadores, backup.

- O antivírus é um programa que detecta e remove o vírus, que é um programa malicioso.
- Emuladores são softwares que simulam, reproduzem um ambiente específico, por exemplo, o VmWare que simula um computador.
- Desfragmentadores eliminam a fragmentação de um sistema. O que é a fragmentação? Pense quando salvamos um arquivo no computador/*smartphone*, e em um outro momento editamos, fazemos algumas modificações naquele arquivo, as modificações não são salvas ao lado do arquivo original. Pense em um armário com várias gavetas: o seu arquivo original está na gaveta 1 e a edição desse arquivo está na gaveta 20; isso ocasiona lentidão na leitura e na gravação. É claro que a explicação técnica é mais complexa que essa abordada aqui, mas esse exemplo serve para entender o que é fragmentação. A desfragmentação ajuda a juntar os arquivos e sua edição próximos uns aos outros.
- Formatadores e backup: a formatação apaga tudo e reinstala.

1.3 Software aplicativo

O software aplicativo é também chamado apenas de aplicativo ou mais popularmente *app*. Podemos dizer que é um software que realiza algum trabalho para o usuário, ou seja, são programas que realizam tarefas específicas. Como

exemplo temos as planilhas, o editor de texto, os navegadores (*browsers*), o editor de imagem, entre outros.

A figura seguinte ilustra os ícones de vários softwares aplicativos.

TEMA 2 – SISTEMA OPERACIONAL

Um sistema pode ser definido como um conjunto de partes que se interagem para alcançar um determinado objetivo.

O que diferencia software de um sistema é que este pode ser definido como um **conjunto de softwares** que interage para alcançar um determinado objetivo.

O sistema operacional faz parte da categoria de **software básico**. Como visto anteriormente, o software básico é aquele necessário para o funcionamento do hardware ou de parte dele. Sem dúvida, sendo o software mais importante, devido às tarefas que desempenha, o sistema operacional é responsável pela supervisão dos processos executados em um computador. O sistema operacional gerencia todo o hardware e todo o software do computador e realiza a "comunicação" entre eles.

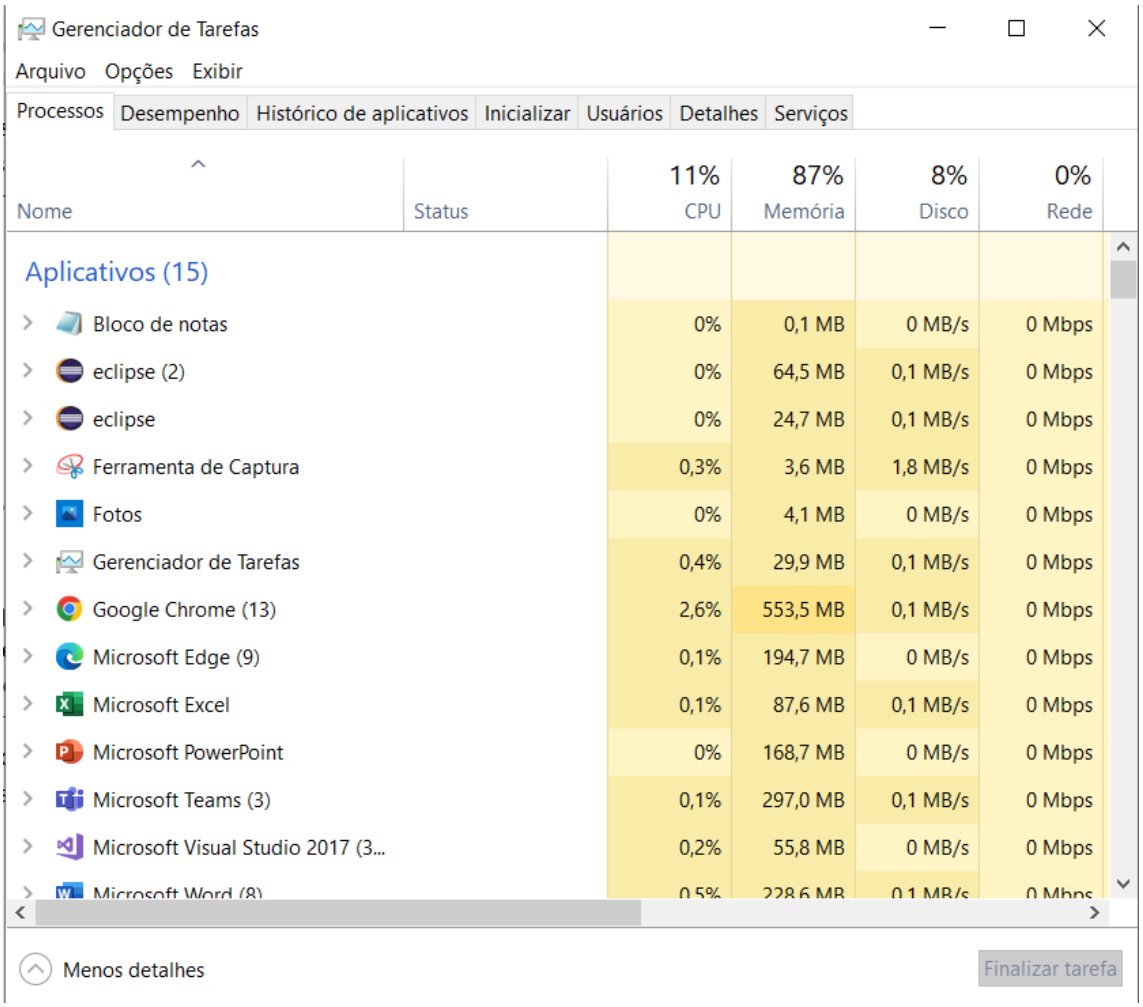
Ao se falar em processo, você pode relacionar com um programa em execução para melhor entendimento. Embora a definição de programa e processos sejam diferentes, neste momento, podemos fazer essa analogia.

Quando você abre o gerenciador de tarefas, você consegue ver vários programas, ou seja, vários processos executando. O sistema operacional vai ser responsável pela execução/supervisão desses processos. Além dos softwares, o SO gerencia o hardware, pois ele é responsável pela comunicação entre o software e o hardware.

A figura seguinte apresenta o gerenciador de tarefas e mostra na aba Processos todos os processos que estão abertos no computador.

Para abrir o gerenciador de tarefas no seu computador, você pode usar o atalho do teclado: CTRL + SHIFT + ESC, ou, ainda, digitar gerenciador de tarefas na barra de pesquisa do Windows. Também, é possível usar CTRL + ALT +DEL e escolher a opção Gerenciador de Tarefas.

Figura 1 – Print do Gerenciador de Tarefas do Windows



Podemos, ainda, dizer que o sistema operacional é uma camada entre o aplicativo utilizado pelo usuário e o hardware. Observe a próxima figura, a qual esquematiza essa representação: a camada de hardware, aqui representada pelo computador, a placa de vídeo, placa mãe, HD, memórias, CPU e periféricos. Veja que temos dois exemplos de SO – Windows e Linux – e alguns aplicativos, como o Chrome, o Firefox e o Word.

Figura 2 – SO como camada intermediária entre usuário e hardware

Crédito: Iamnee/shutterstock.

O sistema operacional desempenha as seguintes funções:

- facilita o uso do computador pelo usuário, tornando mais simples a utilização de seus recursos;
- gerencia os recursos do computador; e
- controla a execução de programas pela CPU.

Imagine se você não tivesse um sistema operacional instalado em seu computador, como faria para utilizá-lo? Se não tiver o SO instalado, você não conseguiria instalar os aplicativos.

Observe que quando falamos em recursos, queremos dizer todo o hardware do computador e, consequentemente, o software. O sistema operacional gerencia todo o hardware, que é a placa de vídeo, placa de rede, memórias e os softwares instalados no computador. Quando você conecta um *mouse*, um *headset* no seu computador, eles também serão gerenciados pelo sistema operacional. Este, por sua vez, ainda controla a execução dos programas pela CPU. A CPU é o processador, é o acrônimo de Central Process Unit. Todo programa que necessita da utilização da CPU é controlado pelo SO.

2.1 Sistemas tradutores

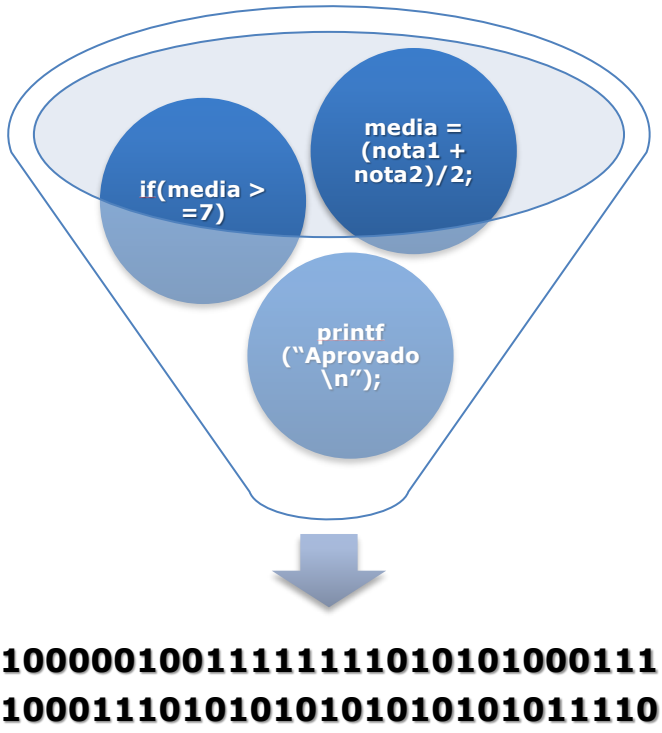
Como integrantes do sistema básico, embora não tenham sido citados anteriormente, temos os sistemas tradutores, que são importantes para conhecermos, principalmente para quem irá trabalhar com desenvolvimento de software.

Os sistemas tradutores convertem os programas escritos para um código em uma linguagem de máquina, mais adequada para manipular *bits*.

Lembra dos códigos escritos nas linguagens de programação? Para que o computador entenda esse código, é necessário que ele seja traduzido para uma linguagem de máquina. Lembre-se de que o computador entende apenas *bits*.

A figura a seguir ilustra um tradutor. O código entra e o tradutor o traduz para uma linguagem de máquina.

Figura 3 – Tradutor



Um programa escrito em linguagem de alto nível tem a necessidade de ser traduzido para a linguagem de máquina para que o computador possa executá-lo.

Temos dois tipos de tradutores: os interpretadores e os compiladores. Então, todo programa escrito em linguagem de alto nível, aqui, quando falamos

em linguagem de alto nível, fazemos referências às linguagens que utilizamos para desenvolvimento de *apps* (aplicativos).

Existem, portanto, linguagens interpretadas e compiladas.

A diferença é que nos interpretadores o código é interpretado linha por linha. Tome como exemplo um intérprete. Pode ser o intérprete de libras, ou outro qualquer, que, enquanto falamos, ele traduz.

Já os compiladores traduzem o código todo de uma vez, gerando um .exe, por exemplo. Como analogia temos a tradução de um livro, em que o tradutor traduz todo o livro e ao final entrega-o traduzido.

Figura 4 – Interpretador

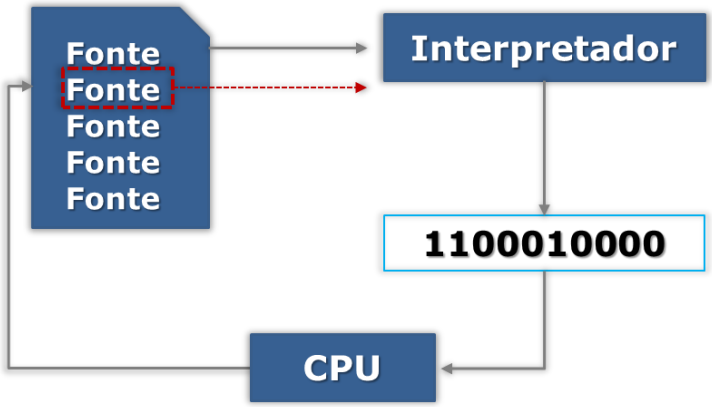
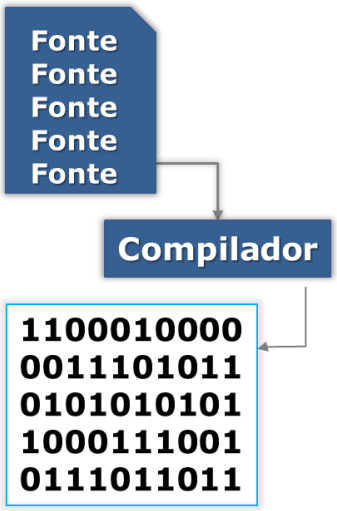


Figura 5 – Compilador



TEMA 3 – ALGORITMOS E LINGUAGENS

No tópico anterior, abordamos os softwares, programas e sistemas. Aqui, iremos conhecer um pouco mais sobre os algoritmos e as linguagens de programação.

3.1 Algoritmos

No tópico anterior, vimos algo dos algoritmos. Foi mencionado que podem assumir que é algo parecido com a receita de bolo. A seguir, apresentamos uma definição de algoritmo. Essa definição é bem simples e sucinta e descreve perfeitamente o que ele é. Tal definição foi dada por Forbellone (2022, p. 3), em um livro sobre lógica de programação bastante utilizado como referência: “Algoritmo é uma sequência de passos que visa atingir um objetivo bem definido”.

Nessa definição de algoritmo, uma sequência passo a passo visa atingir um objetivo bem definido. O objetivo aqui é representar mais fielmente o raciocínio envolvido na lógica de programação. Uma vez concebida uma solução, pode ser traduzida para qualquer linguagem de programação.

Quem já teve Lógica de programação deve ter percebido que existe um *gap* semântico muito grande entre lógica de programação e a lógica do dia a dia. Se você ainda não teve Lógica de programação, quando tiver irá perceber esse *gap*. Quando falamos em semântica na linguagem de programação, estamos nos referindo ao conteúdo, ao significado.

Vamos ver primeiro um exemplo em português para entendermos sintaxe e semântica. A sintaxe está relacionada com as regras, premissas, restrições. De maneira simplificada, as regras que determinam o modo como as palavras podem combinar-se para formar os enunciados definem a sintaxe. Tomemos como exemplo a frase:

```
Fiz prova de FDS na semana passada.
```

Se escrevêssemos:

```
FDS passada semana prova fiz.
```

Ainda assim reconheceríamos as palavras, mas a ordem está estranha!

Semântica está relacionada com o conteúdo, o significado das palavras e sua representação. Por exemplo, um texto pode estar bem escrito (sintaticamente correto), mas sem sentido ou conteúdo (sem início, meio ou fim).

Em linguagem de programação a sintaxe define como a forma ou a estrutura das expressões, dos comandos ou das unidades de programas devem ser, ou seja, ela define se um programa está corretamente escrito em uma determinada linguagem de programação.

Por exemplo, a expressão seguinte está sintaticamente incorreta. Só de olharmos vemos que tem algo estranho. Um parênteses que se abriu e não fechou, um colchete perdido. Números que faltam.

```
(2  ] - /
```

Já a semântica define o significado formal dessas expressões, comandos ou unidades de programas. Ela define o significado da forma de cada trecho de código. Os erros de semântica estão relacionados com a lógica de programação. A próxima expressão está sintaticamente correta na linguagem Python, mas semanticamente não faz sentido somar um número e um caractere – na linguagem python, os caracteres são escritos utilizando aspas duplas (“) ou simples(‘).

```
a = 2 + "3"
```

Voltando ao assunto dos algoritmos, veremos, agora, alguns exemplos. A seguir, é mostrada uma receita de bolo de caneca. O modo de preparo é o passo a passo de como você deve fazer o bolo. No caso, é o algoritmo para se fazer um bolo.

Exemplo 1 de algoritmo

Receita de bolo de caneca

Ingredientes

- 2 colheres (sopa) de achocolatado
- 3 colheres (sopa) de farinha de trigo
- 3 colheres (sopa) de açúcar
- ½ colher (chá) de fermento em pó
- 3 colheres (sopa) de Leite
- 2 colheres (sopa) de óleo
- 1 ovo

Crédito: Africa Studio/shutterstock.

Modo de preparo

Em uma caneca com capacidade superior a 350 ml, misture o achocolatado, a farinha de trigo, o açúcar e o fermento em pó.
Acrescente o leite, o óleo e o ovo e misture delicadamente até incorporar.
Coloque a caneca sobre um prato de sobremesa e leve ao forno micro-ondas por 3 minutos em potência alta.
Sirva a seguir.

Crédito: TownFox/Shutterstock.

O segundo exemplo mostra um algoritmo para calcular a média da nota de duas provas.

Exemplo 2 de algoritmo

Calcular a média de prova

- Obter as duas notas de provas
- Calcular a média aritmética
- Se a média for igual ou maior que 7, o aluno está aprovado
- Se não, está reprovado

Figura 6 – Print de tela de programa: cálculo de média

3.2 Linguagens

Agora falaremos sobre as linguagens de programação. Você se lembra do que falamos sobre algoritmos? Depois que um problema é resolvido, ou melhor, depois que é feito um algoritmo para o problema, o programador passa

esse algoritmo para uma linguagem de programação. O programador é quem profundamente conhece os detalhes da linguagem de programação que ele vai desenvolver. Costumamos dizer que depois de feito um algoritmo, este pode ser implementado em qualquer linguagem de programação.

A seguir, temos uma imagem esquematizando que, após desenvolvermos os algoritmos, que representam a lógica de programação com a estrutura de dados, basta implementar para uma linguagem de programação.

Figura 7 – Esquema: algoritmo + Lógica + ED = Linguagem

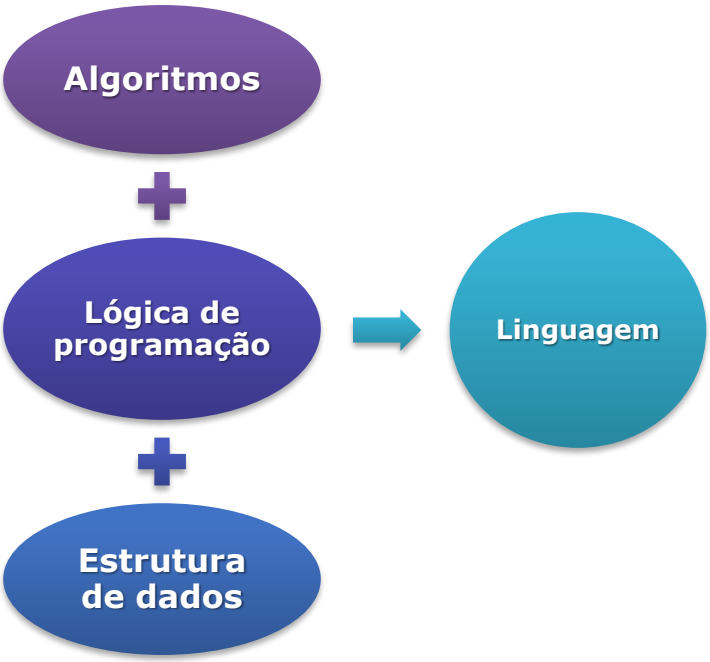
C

C++

C#

Java

Python



É preciso lembrar que cada linguagem de programação tem uma sintaxe específica, com regras gramaticais próprias, como as que temos nas linguagens naturais. Observe nos próximos exemplos a utilização de comando para imprimir

na tela a frase “olá mundo!”. Cada linguagem tem um comando próprio para realizar essa tarefa.

Para imprimir uma frase na tela, em **linguagem C**, podemos fazer:

```
printf(“Olá Mundo!”);
```

Em **linguagem Java**, o comando para imprimir algo na tela seria:

```
System.out.print(“Olá Mundo!”);
```

Em linguagem Python:

```
print(“Olá Mundo!”)
```

No quadro seguinte, podemos visualizar o algoritmo do exemplo 2 implementado em três linguagens de programação diferentes.

Algoritmo implementado na linguagem Python:

```python
#Obter as duas notas de provas
nota1 = float(input("Digite a primeira nota: "))
nota2 = float(input("Digite a segunda nota: "))
#Calcular a média aritmética
media = (nota1 + nota2)/2
#Se a média for igual ou maior que 7, o aluno foi aprovado
if media >= 7:
   print("Aprovado")
#Se não, ele foi reprovado
else:
   print("Reprovado")
```

Algoritmo implementado na linguagem Java:

```java
//Obter as duas notas de provas
      System.out.println("Digite a primeira nota: ");
      float nota1 = teclado.nextFloat();

      System.out.println("Digite a segunda nota: ");
      float nota2 = teclado.nextFloat();

//Calcular a média aritmética
      float media = (nota1 + nota2)/2;

//Se a média for igual ou maior que 7,o aluno foi aprovado,
      if (media >= 7)
            System.out.println("Aprovado");
//Se não, ele foi reprovado
```

```
      else
            System.out.println("Reprovado");
```

Algoritmo implementado na linguagem C:

```
//Obter as duas notas de provas
           printf("Digite a primeira nota: ");
           scanf_s("%d", &nota1);

           printf("Digite a segunda nota: ");
           scanf_s("%d", &nota2);

     //Calcular a média aritmética
           media = (nota1 + nota2) / 2;
     //Se a média for igual ou maior que 7, o aluno foi
aprovado
           if (media >= 7)
                 printf("Aprovado");
     //Se não, ele foi reprovado
              else
                printf("Reprovado");
```

Já a estrutura de dados diz respeito à forma como estes são armazenados e organizados na memória. Como estrutura de dados mais simples temos os vetores, também chamados de *arrays* ou listas. Cada linguagem tem formas e comandos específicos para manipular as estruturas de dados.

TEMA 4 – GLOSSÁRIO DO DESENVOLVEDOR DE SOFTWARE

Toda área tem seu jeito próprio de se comunicar, temos o chamado jargão. Segundo o dicionário Michaellis, “jargão” é a linguagem própria de um grupo profissional, sobretudo no nível lexical; gíria profissional.

Na área da Computação, muito mais que em outras áreas, temos as palavras ou termos que são frequentemente utilizados na área de desenvolvimento. São termos tão corriqueiramente utilizados pelos profissionais da área que é necessário conhecê-los para compreender e conseguir se comunicar com seus pares.

Até há pouco tempo muitas dessas siglas ou jargões eram totalmente desconhecidas pelas pessoas que não eram da área de TI (Tecnologia da Informação). Com a popularização da internet, muitas palavras que eram consideradas como jargão de TI passaram a ser utilizadas por muitas pessoas.

Mas ainda temos alguns acrônimos, palavras ou termos que são utilizados e compreendidos apenas por quem trabalha na área de desenvolvimento. A seguir são citados alguns termos utilizados na área, que são importantes para o profissional de TI conhecer.

Front-end: esse termo está relacionado com desenvolvimento da parte gráfica de uma aplicação web.

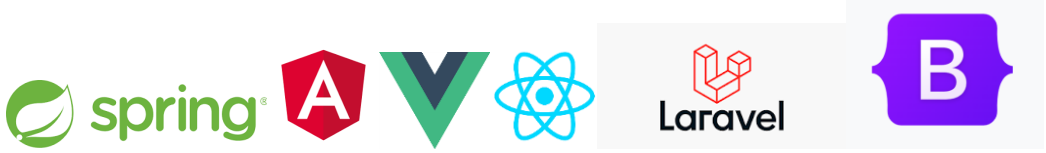
Back-end: está relacionado ao desenvolvimento no lado do servidor. Ou seja, a programação que acontece no lado do servidor. Geralmente, o profissional que trabalha com Back-end tem experiência em versionamento e controle, deve ter, ainda, a habilidade para gerenciar o ambiente de hospedagem, levando em consideração a acessibilidade e segurança.

Full-stack: o desenvolvedor *full-stack* é aquele que trabalha com ambas as abordagens tanto da parte gráfica, quanto da parte do lado do servidor.

API: a *Application Programming Interface* (Interface de Programação de Aplicativos) pode ser definida como um conjunto de rotinas e padrões de programação que possuem o objetivo de acessar aplicativos de software ou plataformas baseados na web. A API é uma interface utilizada por um programa ou aplicação e não por um usuário.

Framework: é um conjunto de código de uma linguagem de programação específica que auxilia no desenvolvimento de projetos de *web* ou de software. Podemos pensar no *framework* como se fosse uma biblioteca de códigos com funções já prontas. Na área de TI, também é comumente vista a definição como arcabouço de código. Exemplo de *frameworks* são: Vue.js, Angular, Bootstrap e React *frameworks* para desenvolvimento *front-end*. Spring boot e Laravel: *framework* para aplicações *web back-end*.

A seguir temos logotipos do *spring*, angular, vue.js, *react*, *laravel* e *bootstrap*, respectivamente.



IDE: é o acrônimo para *Integrated Development Environment*. O ambiente de desenvolvimento integrado é um software que integra diversas funcionalidades para desenvolvimento em uma única interface gráfica. A IDE

auxilia e agiliza o processo de desenvolvimento. São exemplos de IDE: Pycharm, Eclipse, Apache Netbeans, Visual Studio.

SDK: acrônimo para *Software Development Kit*, em português: kit de desenvolvimento de software. Basicamente, um SDK será composto de compilador, *debugger* e API. Um SDK pode ser entendido como conjunto de ferramentas fornecidos por um fabricante para que se desenvolva para uma plataforma ou sistema específico.

Nativo: um aplicativo nativo é aquele desenvolvido para uma única plataforma utilizando linguagens e ferramentas específicas para a plataforma em questão. Por exemplo, para desenvolvimento nativo no Android, fazemos uso de uma SDK (Android SDK) e uma IDE (Android Studio), utilizando a linguagem Java e kotlin.

Híbridos: dizer que uma aplicação é híbrida implica falar que sua implementação utiliza html, css e Javascript. As aplicações híbridas utilizam *frameworks* ou ferramentas que permitem uma mesma base de código. A aplicação híbrida é desenvolvida em uma única linguagem e distribuída para várias plataformas.

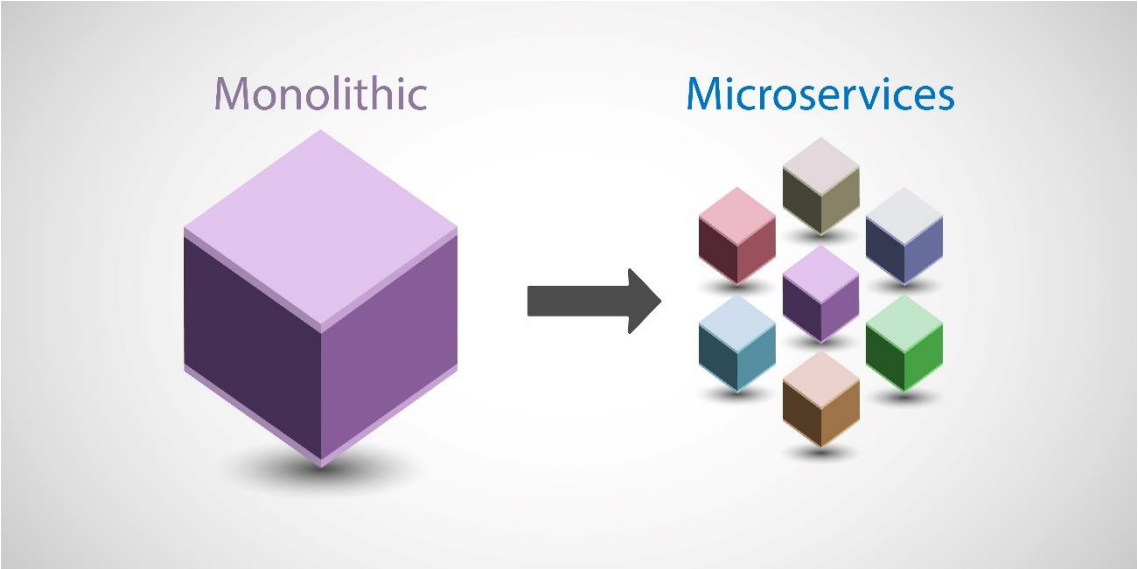
Serviços: na área de desenvolvimento podem ser entendidos como processos de software. O termo processo de software, nesse caso, não é no sentido de conjunto de atividades que temos na engenharia de software, mas em um programa processado pelo SO (Sistema Operacional).

Monolítico: é uma aplicação que roda com um único processo.

Microsserviço: é uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes que se comunicam usando APIs bem definidas. Esses serviços pertencem a pequenas equipes autossuficientes. Como características dos microsserviços temos que eles são autônomos e especializados.

A próxima figura ilustra o conceito de aplicação monolítica e microsserviços.

Figura 8 – Aplicação monolítica x microsserviços



Crédito: Ashalatha/shutterstock.

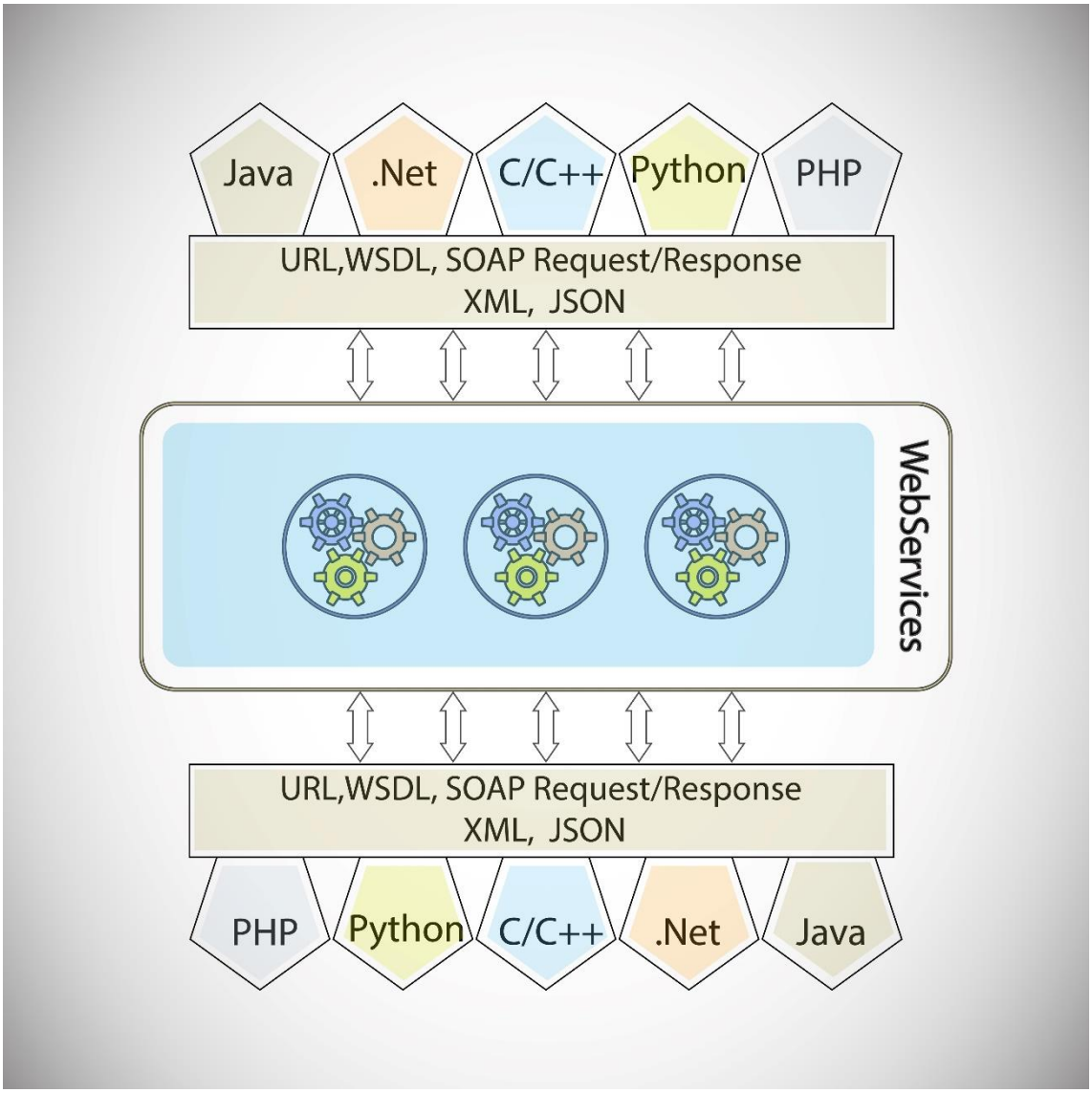
SOAP: *Service-Oriented Architecture* (*Or Application*) *Protocol* (protocolo de arquitetura orientada a serviços). O SOAP utiliza arquivos xml e o protocolo HTTP como protocolo de transporte.

REST: *REpresentational State Transfer*, em português: Transferência Representacional de Estado, é um conjunto de restrições para criação de *webservices*. Quando um serviço implementa esse padrão, dizemos que é Restfull. Restfull utiliza arquivos JSON.

SOAP e REST são APIs e são arquiteturas de *webservice*. *Webservices* são independentes de linguagem. Segundo Ferreira (2021), é correto afirmar que todo *Webservice* é uma API, mas nem toda API é um *Webservice*, porque tanto o *Webservice* quanto as APIs realizam a comunicação entre aplicações, porém a forma como são utilizados é totalmente diferente.

A figura seguinte ilustra a independência dos *webservices* em relação às linguagens.

Figura 9 – Independência de *webservices* em relação a linguagens de programação



Crédito: Ashalatha/Shutterstock.

Commit: o termo *commit* ou *"comitar"* significa enviar alterações de um determinado trecho do código, ou seja, o envio da criação de uma nova versão do projeto. Geralmente, quando o desenvolvedor termina uma tarefa, ele gera um *commit*. Esse *commit* possui o código modificado e a explicação do que esse *commit* faz naquele código.

Versionamento: consiste em atribuir um número de versão ao estado do projeto. À medida que o desenvolvedor codifica, ele vai criando versões do projeto. Tome como exemplo várias pastas que representam versões do seu projeto: trabalho_FDS_v1, trabalho_FDS_v2.

Snapshot: cópia instantânea em um determinado tempo de um volume.

Git: é um sistema de controle de versão que gerencia as várias versões no desenvolvimento de um documento. Criado por Linus Torvalds, a justificativa para a escolha do nome é controversa, sendo que nenhuma delas foi desmentida pelo autor. GIT, em inglês britânico, é uma gíria para cabeça dura, também pode ser o acrônimo para "*Global Information Tracker*" ou, ainda, "*Goddamn idiotic truckload of sh*t*" (é um xingamento, não iremos traduzir). O logo do GIT representa a ramificação para desenvolvimento não linear.

Github: segundo a própria desenvolvedora, o Github é uma plataforma de desenvolvedor completa para criar, dimensionar e fornecer software seguro. O github utiliza o Git como sistema de controle.

Há uma certa confusão entre Git e GitHub para quem começa na área. Entenda Git como uma ferramenta. Tendo esta, você ainda precisará de um servidor, que, no caso, é o Github. Embora seja mais conhecido e usado, também existem outros servidores que prestam suporte para git, como: Gitlab, Bitbucket, Apache Allura. Os ícones seguintes são, respectivamente, do GitHub, Gitlab, Bitbucket e Apache Allura.

Debug: o *debugging* ou debugar significa depurar o programa. Ou seja, encontrar erros e tentar resolvê-los. Aqui, só lembrando que o termo *bug* na área de TI se refere a um erro no programa.

TEMA 5 – CICLO DE VIDA DE SOFTWARE

O ciclo de vida de um software (do inglês: *Software Development Life Cycle* – SDLC), também chamado de modelo de processo, é uma representação simplificada de um processo de software. Este é como se fosse um roteiro, uma série de passos previsíveis. Esse processo depende do software que está sendo

desenvolvido. O ciclo de vida de um software indica as etapas que devem ser cumpridas e a sequência para que o software seja desenvolvido.

Os modelos de processos trazem ordem ao caos existente na área de desenvolvimento de software. O engenheiro de software deve adaptar um modelo de processo às suas necessidades e, então, deve segui-lo. O processo dá controle, estabilidade e organização ao desenvolvimento de software.

Segundo Sommerville (2019), quatro atividades são comuns a todos os processos de software: especificação, desenvolvimento, validação e evolução.

Antes de explicar mais detalhadamente cada fase, para ficar fácil de entender, podemos fazer uma analogia com a construção de uma casa. Essas fases genéricas são semelhantes também às fases da construção de uma moradia; a fase especificação, por exemplo, seria especificar com o contratante como ele quer que a casa seja. Na fase de desenvolvimento, é realizada a construção da moradia propriamente dita. Na fase de validação, é garantido que a casa está sendo construída de acordo com o que o contratante solicitou. A fase de evolução serve para manter a casa em boas condições, ou seja, evoluir para atender às necessidades do contratante. É possível incluir: pintura, troca/retirada/acréscimo de encanamento, troca/retirada/acréscimo de fiação, conserto de problemas que surgirem após final da construção, construção de um novo cômodo etc.

Entendido o que cada fase faz na construção da casa, vejamos essa definição para o desenvolvimento de software.

Na fase de especificação de um software ocorre a definição do problema. A funcionalidade e as restrições quanto às funções do software são definidas aqui. Nessa fase, é feita uma reunião com o cliente para levantamento de requisitos. Existem basicamente dois tipos de requisitos: os funcionais e os não funcionais. Os primeiros dizem respeito ao que o software deve fazer, ou seja, quais funções ele deve possuir. Já os não funcionais dizem respeito às restrições do software.

Na etapa de desenvolvimento de software ocorre a codificação do software de acordo com as especificações coletadas na fase anterior. Aqui, também é feita a escolha da linguagem de programação que será utilizada para implementação do sistema.

Na fase de validação o programa é analisado para garantir que atenda ao que foi solicitado pelo cliente.

Na fase de evolução o software deve evoluir de acordo com a mudança nas necessidades do cliente, ou seja, devem ser realizados ajustes caso o software ou algum módulo necessite de alguma modificação.

5.1 Ciclo de vida de software (SDLC) – metodologia tradicional

Existem muitos modelos de processo de software. A seguir é mostrado um modelo clássico conhecido como modelo cascata (*waterfall*). O modelo é mostrado em uma perspectiva arquitetural; isso significa que será mostrada sua estrutura, mas não os detalhes das atividades. Devido à tradicionalidade do modelo, muitas vezes, ele é referenciado como sendo o próprio SDLC.

Modelo Cascata (*Watterfall*): mostra as atividades fundamentais do processo de software distribuídas em fases distintas. Devido à cascata de uma fase para outra, é definido como modelo cascata ou ainda conhecido como **modelo de ciclo de vida do software (SDLC)**.

São cinco fases que compõem o modelo cascata: requerimento, projeto, implementação, verificação e manutenção.

Na etapa de requerimento ocorre o levantamento de requisitos. As metas e restrições são identificadas com os usuários.

Na etapa de projeto os requisitos identificados na etapa anterior são mapeados em componentes de hardware e software.

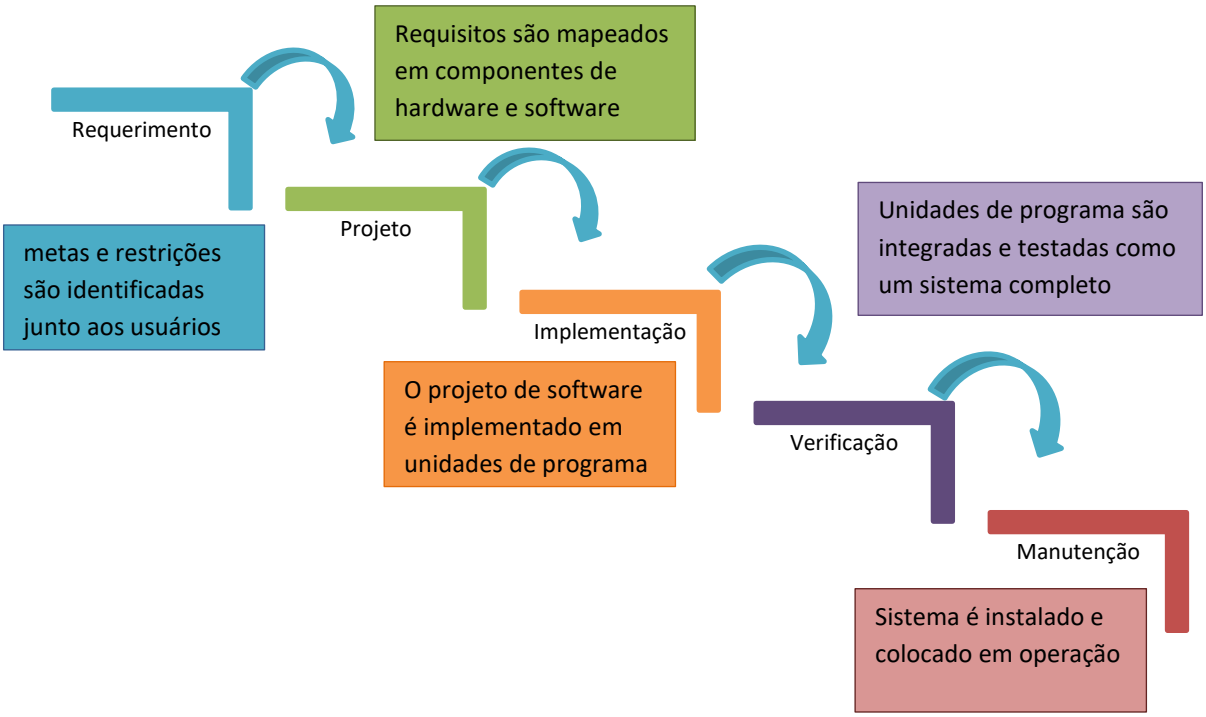
Na etapa de implementação, o projeto de software é implementado em unidades de programas. Nessa fase ocorre também o teste de unidade, que envolve a verificação de cada unidade e se cada uma cumpre a sua especificação.

Na etapa de verificação as unidades de programas são integradas e testadas como um sistema completo.

Na etapa de manutenção o sistema é instalado e colocado em operação. Aqui, são corrigidos os erros que não foram descobertos nas fases anteriores. Novas funcionalidades podem ser identificadas e se há a necessidade da repetição do ciclo.

A figura a seguir representa o modelo cascata e a descrição sucinta de suas fases.

Figura 10 – Modelo cascata (*Waterfall*)



Fonte: Kanashiro, 2022.

O modelo cascata é fácil de gerenciar, pois tem etapas bem definidas e sem sobreposição. É um modelo que visa à alta qualidade, enfatizando metas e pontos de revisão. Porém, esse modelo apresenta desvantagens, pois é improdutivo quanto ao tempo, já que a etapa posterior só pode iniciar depois da finalização da etapa anterior. A Visão Sequencial também não corresponde ao mundo real, fornecendo pouca visibilidade do estado do projeto, pois é demandado muito tempo para a primeira entrega. Existem vários outros processos de software, porém o Cascata é o mais conhecido e tradicional.

5.2. Ciclo de vida de software – metodologia ágil

A metodologia ágil é uma alternativa para a gestão de projetos tradicionais.

Em 2001, um grupo de programadores lançou o Manifesto Ágil, pregando uma metodologia que tem como objetivo satisfazer os clientes entregando com rapidez e com maior frequência versões do software conforme suas necessidades. A metodologia ágil entrega as funcionalidades de software mais

rapidamente aos seus clientes. Essa metodologia tem foco no software e não no projeto ou na documentação. É adequada para aplicações em que os requisitos mudam rapidamente. Veremos duas abordagens utilizadas na metodologia Ágil: o *extreme Programming* e o SCRUM.

5.3 XP (*eXtreme Programming*)

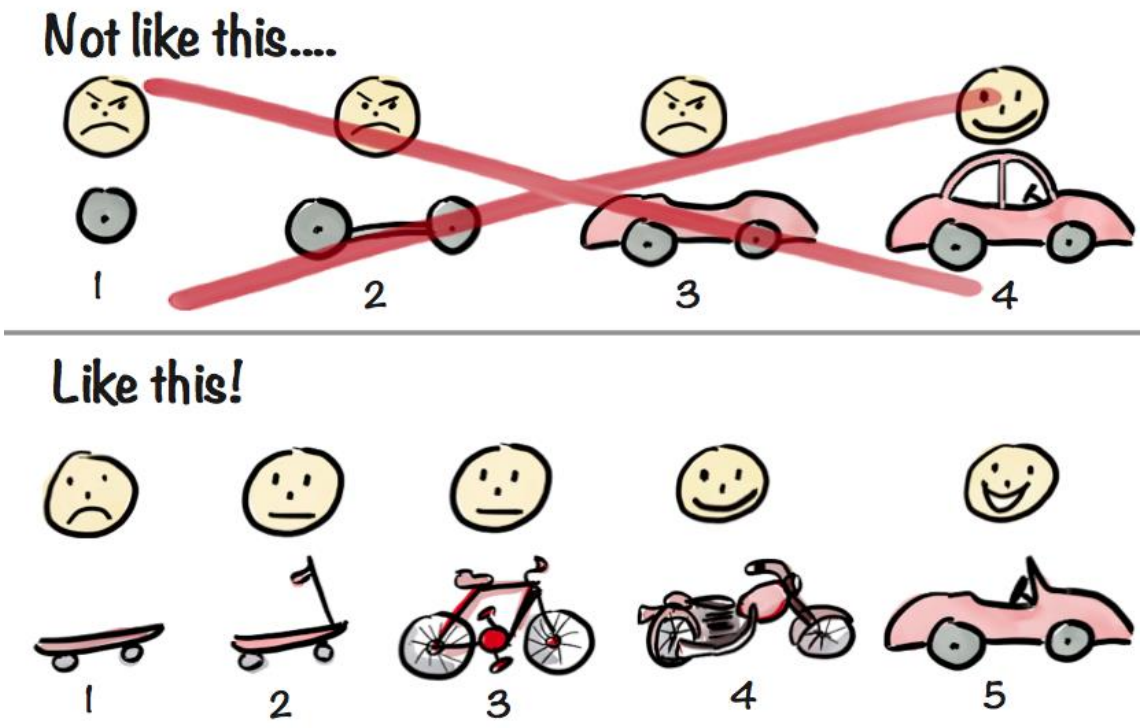
A programação extrema introduziu práticas ágeis ao desenvolvimento tradicional de software. As práticas da XP refletem os princípios do manifesto ágil. O quadro seguinte descreve as práticas do XP e suas respectivas descrições.

Quadro 1 – Práticas do XP e suas respectivas descrições

Princípio ou prática	Descrição
Propriedade coletiva	Os pares de desenvolvedores trabalham em todas as áreas do sistema de modo que se desenvolvem "ilhas de conhecimento", e todos os desenvolvedores assumem a responsabilidade por todo o código. Qualquer um pode mudar qualquer coisa.
Integração continua	Assim que o trabalho em uma tarefa é concluído, ele é integrado ao sistema completo. Após qualquer integração desse tipo, todos os testes de unidade no sistema devem passar
Planejamento incremental	Os requisitos são registrados em "cartões de história, e as histórias a serem incluídas em um lançamento são determinadas de acordo com o tempo disponível e com sua prioridade relativa. Os desenvolvedores decompõem essas histórias em tarefas de desenvolvimento.
Representante do ciente	Um representante do usuário final do sistema (o cliente) deve estar disponível em tempo integral para o time de programação. Em um processo como esse o cliente é um membro do time de desenvolvimento, sendo responsável por levar os requisitos do sistema ao time, visando sua implementação
Programação em pares	Os desenvolvedores trabalham em pares, conferindo o trabalho um do outro e oferecendo apoio necessário para que o resultado seja sempre satisfatório
Refatoração	Todos os desenvolvedores devem refatorar o código continuamente logo que sejam encontradas possíveis melhorias para ele. Isso mantém o código simples e de fácil manutenção
Projeto(design) simples	Deve ser feito o suficiente de projeto(design) para satisfazer os requisitos atuais e nada mais.
Lançamentos pequenos	O mínimo conjunto útil de funcionalidade que agregue valor ao negócio é desenvolvido em primeiro lugar. Os lançamentos do sistema são frequentes e acrescentam funcionalidade à primeira versão de uma maneira incremental.
Ritmo sustentável	Grandes quantidades de horas extras não são consideradas aceitáveis, já que o efeito líquido muitas vezes é a diminuição da qualidade do código e da produtividade no médio prazo
Desenvolvimento com testes a priori(test first)	Um framework automatizado de teste de unidade é utilizado para escrever os testes de um novo pedaço de funcionalidade antes que ela própria seja implementada.

Fonte: Sommerville, 2018, p. 63.

A figura seguinte ilustra metaforicamente a metodologia tradicional que entrega apenas parte do produto *versus* a metodologia ágil, onde pequenas versões do projeto são entregues até que se entregue o produto.



Crédito: Henrik Kniberg.

5.4 Scrum

O manifesto ágil possibilitou o aparecimento de várias metodologias e *frameworks*. Um dos mais utilizados é o scrum. E é sobre ele que veremos a seguir.

O Scrum é um *framework* estrutural, não é considerado uma metodologia. Basicamente, uma metodologia diz o que fazer e como fazer. O Scrum indica uma trajetória, mas não como fazer. Sendo assim, o Scrum é um arcabouço de informações, um esqueleto.

O Scrum possui três artefatos, cinco eventos, cinco valores e três papéis. A figura a seguir mostra os respectivos artefatos, eventos, valores e papéis do Scrum. Vamos ver um pouco mais sobre essa composição do Scrum de acordo com o seu próprio guia. Alguns termos serão mantidos em inglês, pois são as terminologias utilizadas na área.

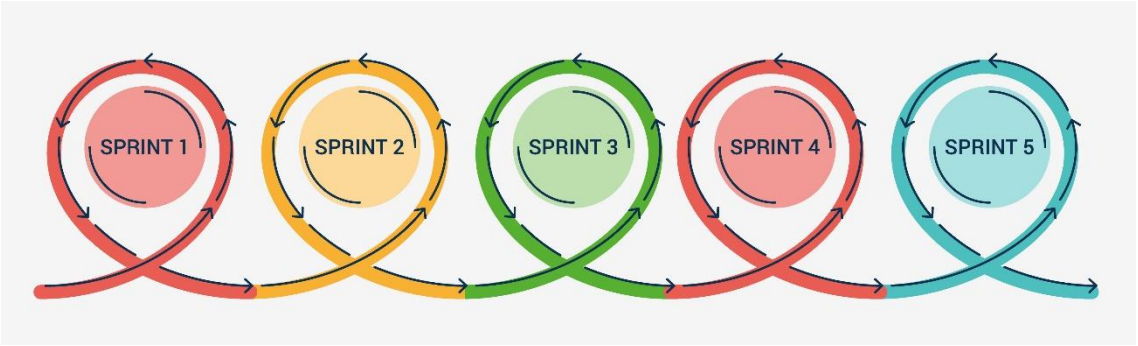
Figura 11 – Componentes do Scrum

Fonte: Kanashiro, 2022.

No Scrum, os projetos são divididos em ciclos chamados de Sprints. As Sprints são a essência, o coração do Scrum. Uma sprint é um *time-boxed* de um mês ou menos, durante o qual um "Pronto", incremento de produto potencialmente liberável é criado. O *time-boxed* é a expressão para definir uma unidade de tempo no Scrum. Cabe, ainda, ressaltar que as Sprints têm durações consistentes ao longo de todo o desenvolvimento. Uma nova Sprint só se inicia imediatamente após a conclusão da Sprint anterior. As sprints podem ser vistas como as iterações que ocorrem ao longo do desenvolvimento de software.

A próxima figura ilustra a consistência das Sprints.

Figura 12 – Sprints



Crédito: KPDMedia/Shutterstock.

Artefatos: no Scrum representam o trabalho feito para fornecer transparência ao processo. São projetados para maximizar a transparência das informações-chave para que todos no time recebam as informações relevantes e tenham a mesma compreensão do que deve ser feito.

Como colocado anteriormente, o Scrum possui três artefatos: o *product backlog*, o *sprint backlog* e o *increment*.

O ***Product Backlog*** é uma lista ordenada de tudo que é conhecidamente necessário para fornecer o produto. É a única origem dos requisitos para qualquer mudança a ser feita no produto. O *product backlog* é, então, uma lista de produtos pendentes.

O ***Sprint backlog*** possui os itens do *product Backlog* selecionados para a Sprint, com o plano para entregar o incremento do produto e atingir o objetivo da Sprint. Possui também uma previsão do Time de Desenvolvimento sobre qual funcionalidade estará no próximo incremento. O Sprint Backlog é um plano feito por e para desenvolvedores. É uma imagem em tempo real altamente visível do trabalho que os desenvolvedores planejam realizar durante o Sprint para atingir o objetivo do Sprint.

O ***increment*** (incremento) é a soma de todos os itens do *Product Backlog* completados durante a Sprint e o valor dos incrementos de todas as Sprints anteriores.

Eventos: o Scrum prescreve alguns eventos formais para inspeção e adaptação:

- Sprint;
- Sprint Planning (Planejamento da Sprint);
- Daily Scrum (Reunião diária);
- Sprint Review (Revisão da Sprint); e
- Sprint Retrospective (Retrospectiva da Sprint).

O Sprint foi explicado previamente. Iniciaremos, então, com a conceituação do **Sprint Planning** ou o Planejamento da Sprint. O **Planejamento do Sprint** inicia o Sprint, estabelecendo o trabalho a ser realizado para o Sprint. Esse plano resultante é criado pelo trabalho colaborativo de todo o Time Scrum. O Product Owner garante que os participantes estejam preparados para discutir os itens mais importantes do Product Backlog e como eles são mapeados para a meta do produto. O Time Scrum também pode convidar outras pessoas para participar do Sprint Planning para fornecer conselhos.

O objetivo do **Daily Scrum** é inspecionar o progresso em direção ao Sprint Goal e adaptar o Sprint Backlog conforme necessário, ajustando o próximo trabalho planejado. O Daily Scrum é uma reunião de aproximadamente 15 minutos para os Desenvolvedores do Time Scrum.

A **sprint Review** (Revisão do Sprint) tem como objetivo inspecionar o resultado do Sprint e determinar futuras adaptações. O Time Scrum apresenta os resultados de seu trabalho para as principais partes interessadas e o progresso em direção ao Objetivo do Produto é discutido.

Sprint Retrospective (Retrospectiva da Sprint) tem como objetivo planejar maneiras de aumentar a qualidade e a eficácia. O Time Scrum inspeciona como foi o último Sprint em relação aos indivíduos, interações, processos, ferramentas e sua Definição de Pronto. Aqui, ainda, é discutido o que correu bem durante o Sprint, quais problemas foram encontrados e como esses problemas foram (ou não) resolvidos.

Valores: os cincos valores descritos a seguir sempre foram considerados como partes do Scrum e adicionados oficialmente em julho de 2016 ao guia do Scrum.

- **Coragem**: o time Scrum precisa ter coragem para fazer a coisa certa e trabalhar em problemas difíceis.
- **Foco**: todos focam no trabalho da Sprint e nos objetivos do time Scrum.
- **Comprometimento**: as pessoas se comprometem pessoalmente em alcançar os objetivos do *time scrum*.
- **Respeito**: os membros do *time scrum* respeitam uns aos outros para serem pessoas capazes e independentes.
- **Abertura**: o *time scrum* e seus *stakeholders* concordam em estarem abertos a todo o trabalho e aos desafios com a execução dos trabalhos.

Papéis: a unidade fundamental do Scrum é o time. Um pequeno time, ou seja, um pequeno grupo de pessoas que trabalham juntas, definidas de time Scrum. Não existe o conceito de hierarquia no time Scrum, mas cada integrante do time tem um papel (role). São três papeis: *Product Owner*, *Scrum master* e *developers* (desenvolvedores).

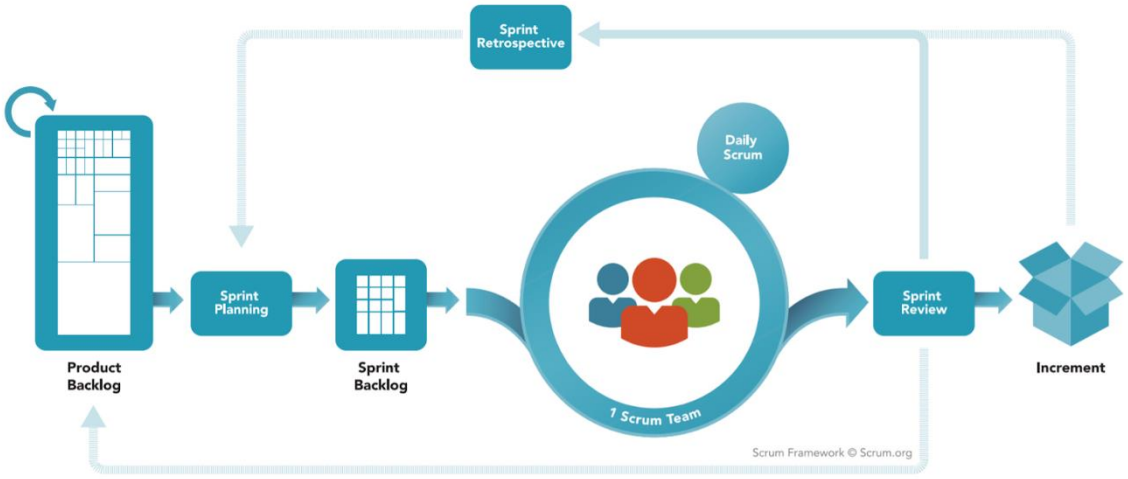
O ***Product owner*** é dono do produto, é o responsável por maximizar o valor do produto resultante do trabalho do Time de Desenvolvimento. É a única pessoa responsável por gerenciar o *Backlog* do Produto.

O ***Scrum master*** é o responsável por promover e suportar o Scrum como definido no Guia Scrum. O *Scrum Master* faz isso ajudando a todos a entenderem a teoria, as práticas, as regras e os valores do Scrum.

O **Time de Desenvolvimento** consiste em profissionais que realizam o trabalho de entregar um incremento potencialmente liberável do produto ao final de cada Sprint.

A figura a seguir representa o ciclo do Scrum.

Figura 13 – Ciclo do Scrum



Fonte: Scrum.org.

FINALIZANDO

Nesta etapa, você conheceu a definição técnica de software e agora já é capaz de diferenciar softwares, programas e sistemas. Conheceu, também, o conceito de algoritmos e alguns exemplos de implementação em algumas linguagens de programação. Foi visto também alguns dos termos mais utilizados pelos desenvolvedores de software e pudemos explorar um pouco sobre o ciclo de vida de software, conhecendo o modelo cascata, modelo ágil e suas características.

REFERÊNCIAS

AWS. AWS Amazon, 2022. **O que são Microsserviços**. Disponível em: <https://aws.amazon.com/pt/microservices/#:~:text=Microsservi%C3%A7os%20s%C3%A3o%20uma%20abordagem%20arquitet%C3%B4nica,pertencem%20a%20pequenas%20equipes%20autossuficientes>. Acesso em: 18 set. 2022.

FORBELLONE, A. L. V. **Lógica de Programação**. 4. ed. Porto Alegre: Bookman, 2022.

MARÇULA, M.; PIO, A. B. F**. Informática - conceitos e aplicações**. 5. ed. São Paulo: Editora Érica, 2019.

MARCONI, M. de A.; LAKATOS, E. M. **Fundamentos de metodologia científica**. 6. ed. São Paulo: Atlas, 2005.

MONTEIRO, E. R. *et al*. **DevOps**. Revisão técnica: FAGONDE, F. S. S.; DIAS, C. de O. Porto Alegre: Sagah, 2021.

PRESSMAN, R. S.; MAXIM, B. R**. Engenharia de software**. Tradução de COSTA, F. A. da; Revisão técnica: ARAKAKI, R.; ARAKAKI, J.; ANDRADE, R. M. de. 9. ed. Porto Alegre: Grupo A, 2021.

SCHWABER, K.; SUTHERLAND, J. **Scrum Guide**: The Definitive Guide to Scrum: The Rules of the Game, 2020.

SOMMERVILLE, I. **Engenharia de Software**. Tradução de BOSNIC, I.; GONÇALVES, K. G. de O. Revisão técnica de Kechi Hirama. 10. ed. São Paulo: Pearson Education do Brasil, 2018.