UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO

Cleyton Vidal Ananias, 2015564
João Carlos da Silva Brito, 2008104
Leandro Pereira, 2015448
Lucas Bezerra de Macedo, 2003247
Raul Segundo Fernandes, 2000956
Thais de Macedo Costa, 2006273

Desenvolvimento de sistema para auxílio na concepção e compartilhamento de planos de aula

São Paulo - SP
2021

UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO

Desenvolvimento de sistema para auxílio na concepção e compartilhamento de planos de aula

Relatório Técnico-Científi-fico apresentado na disciplina de Projeto Integrador para o curso do Eixo Computação da Universidade Virtual do Estado de São Paulo (UNIVESP).

São Paulo - SP
2021

ANANIAS, Cleyton Vidal Ananias; BRITO, João Carlos da Silva; COSTA, Thais de Macedo; FERNANDES, Raul Segundo; MACEDO, Lucas Bezerra de; PEREIRA, Leandro. **Desenvolvimento de sistema para auxílio na concepção e compartilhamento de planos de aula.** 75f. Relatório Técnico-Científico. Eixo Computação – **Universidade Virtual do Estado de São Paulo**. Tutora: Gláucia Uesugi Polo Tiquatira, 2021.

RESUMO

O plano de aula, desenvolvido individualmente por cada um dos docentes, é uma ferramenta essencial para atingir o êxito em um processo de ensino e aprendizagem. Ele retrata um sequenciamento lógico de como transmitir conhecimentos, prover reflexões, propor ações educativas e desenvolver eventuais competências, habilidades e valores. Este trabalho, tem como objeto de estudo o desenvolvimento de um sistema *Web* para auxiliar os docentes da instituição Senac São Paulo (unidade Guarulhos) na construção de plano de aula mais assertivo e que contribua para a aplicação de situações de aprendizagem que mais se adequem as demandas e ao perfil dos discentes, considerando o histórico e o compartilhamento de boas práticas já desenvolvidas e implementadas pela equipe de trabalho. Para prover determinada solução, as bases para o desenvolvimento desta pesquisa são as consultas a materiais bibliográficos pertinentes ao tema; assim como o entendimento das ações pedagógicas da instituição a ser estudada, por meio de seus documentos oficiais em que apresenta sua proposta pedagógica e a utilização de tecnologias para desenvolvimento do sistema e respectiva base de dados, neste caso o *framework* Django e o MySQL, respectivamente. Para melhor entendimento do cenário existente, optou-se pela identificação de requisitos mínimos para desenvolvimento da solução por meio de entrevistas com os agentes do processo educacional (docentes, coordenadores e demais membros da área pedagógica da instituição) e pela prototipação para posterior validação, aperfeiçoamento e/ou correções do recurso a ser desenvolvido.

PALAVRAS-CHAVE: planejamento, plano de aula, educação, solução, Django, MySQL, desenvolvimento, sistema, *web*.

LISTA DE ILUSTRAÇÕES

LISTAS DE TABELAS

LISTAS DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

O plano de aula, desenvolvido pelo docente é um trabalho individual, que consiste no desdobramento de uma atividade realizada anteriormente em conjunto com outros professores que também ministraram aula em um mesmo curso: o Plano Coletivo de Trabalho Docente (PCTD).

Por meio do plano de aula, o docente organiza quais situações de aprendizagem serão ofertadas a uma turma para que ocorra o desenvolvimento de competências, habilidades e valores inerentes ao curso. Em suma, observa-se a relevância deste procedimento para que as ações educacionais sejam concebidas com maior assertividade.

Entretanto, o desenvolvimento do plano de aula, na instituição a ser estudada ao longo do projeto (Senac – Unidade Guarulhos), é realizado sem um formato padrão e este fator contribui para a ausência de um grau maior de detalhamento em alguns elementos importantes que podem resultar em uma situação de aprendizagem, avaliação ou feedbacks menos efetivos. O modelo concebido atualmente, caracteriza-se por um documento desenvolvido pelo próprio docente que é entregue a área técnica pedagógica e não é compartilhado com os demais membros da equipe (docentes), que também ministram aulas para a mesma turma ou atuam em cursos similares.

Diante desse cenário, optou-se por construir um sistema para auxílio ao docente na concepção de plano de aula. A estruturação do plano, via sistema, ocorrerá com base nos elementos presentes na documentação oficial da instituição como competência a ser desenvolvida, critérios de avaliação, tempo para cada situação de aprendizagem, recursos utilizados, critérios de avaliação, entre outros. A solução também permitirá o compartilhamento do plano de aula à toda equipe de docentes que atua no mesmo curso e área, com o intuito de formar um banco de situações de aprendizagem para eventuais aperfeiçoamentos e difusão de boas práticas, assim como um histórico acerca das ações já implementadas.

Para optar pelo tema a ser desenvolvido neste trabalhado foram considerados e analisados, os seguintes critérios:

- grau de aderência ao tema gerador;
- complexidade do projeto, considerando o tempo para desenvolvimento e implementação;
- acesso a dados/informações relevantes para concepção da solução;
- vivência e experiência da equipe em relação aos temas propostos;
- flexibilidade e agilidade a requisições para obtenção de dados e validações;

- disponibilidade de amostra de comunidade para validações e adequações.

A solução concebida, será fruto de estudos e mapeamentos prévios acerca dos seguintes elementos:

- identificação dos elementos essenciais que, obrigatoriamente, constituem o plano de aula;
- seleção e leitura do material bibliográfico pertinente correlacionado ao tema;
- pesquisas e testes com framework e banco de dados selecionados (Django e MySQL, respectivamente);
- prototipação da solução, para posterior validação e/ou aperfeiçoamentos;
- interação por meio de entrevistas com agentes educacionais da instituição (docentes, colaboradores do setor pedagógico e outros).

2. DESENVOLVIMENTO

2.1 Objetivos

Esse projeto tem como objetivo a construção de um sistema para auxílio ao docente na concepção de plano de aula. A estruturação do plano ocorrerá com base no registro e correlação dos seguintes elementos: identificação de curso, identificação de turma, competência a ser desenvolvida, título da situação de aprendizagem, conjunto de atividades que constituem a situação de aprendizagem, duração de cada etapa da situação de aprendizagem, correlação de conhecimentos, habilidades e valores relacionados às atividades, indicadores utilizados para evidenciar o desenvolvimento de competências, critérios de avaliação, recursos utilizados, contribuições para o projeto integrador desenvolvido ao longo do curso, registro de históricos e adequações/correções existentes durante aplicação prática.

A solução também permitirá o compartilhamento do plano de aula à toda equipe de docentes que atua no mesmo curso e área, com o intuito de formar um banco de situações de aprendizagem para eventuais aperfeiçoamentos e difusão de boas práticas, assim como um histórico acerca das ações já implementadas. Para alcançar êxito no desenvolvimento da solução proposta serão necessárias as seguintes etapas ao longo da concepção do projeto:

- estruturar um plano de ação com eventuais prazos e atribuições para posterior controle e monitoramento;
- pesquisar bibliografia correlacionada ao tema abordado e recursos provenientes para o desenvolvimento do sistema proposto;
- elaborar roteiro de entrevistas com agentes envolvidos (docentes, coordenadores e demais colaboradores que serão beneficiados pela solução proposta), com o intuito de especificar demandas e revisar o plano de ação previsto;
- construir diagrama de contexto (visão de alto nível do sistema);
- conceber o DER (Diagrama de Entidade – Relacionamento) e consequentemente identificar tabelas e posteriores normalizações;
- implementar SGBD (Sistema de Gerenciamento de Banco de Dados) e o banco de dados;
- desenvolver solução com o *framework* escolhido;

- validar junto aos agentes envolvidos, por meio de testes, e executar posteriores ajustes ou aperfeiçoamentos;
- analisar resultados obtidos e documentar.

2.2. Justificativa e delimitação do problema

O plano de aula é uma ferramenta essencial para que o processo de ensino e aprendizagem tenha êxito em que instituição educacional. Além de conter as diretrizes propostas para a concepção de uma unidade curricular, o plano de aula, ao ser analisado durante a após sua execução pode ser uma rica ferramenta para entendimento das práticas que foram mais assertivas e melhor compreensão do perfil de uma um grupo de estudantes. A magnitude destas informações pode ser potencializada, caso ocorra o compartilhamento de práticas e percepções a todos os docentes atuantes em um mesmo curso.

O desenvolvimento do plano de aula, na instituição a ser estudada ao longo do projeto (Senac – Unidade Guarulhos), é realizado sem um formato padrão e este fator contribui para a ausência de um grau maior de detalhamento em alguns elementos importantes que podem resultar em uma situação de aprendizagem, avaliação ou feedbacks menos efetivos. O modelo concebido atualmente, caracteriza-se por um documento desenvolvido pelo próprio docente que é entregue a área técnica pedagógica e não é compartilhado com os demais membros da equipe (docentes), que também ministram aulas para a mesma turma ou atuam em cursos similares.

Também em função do elevado número de docentes no quadro da instituição (pouco mais de 50 colaboradores) e as diferentes escalas de horário, o intercâmbio de boas práticas e o compartilhamento de cases que obtiveram um resultado significativo, estão restritos às reuniões pedagógicas ou encontros esporádicos realizados pela instituição.

Em relação a uma turma de um determinado eixo/segmento, observa-se que a troca de informações a respeito do aproveitamento individual e coletivo é realizada mediante as reuniões mensais de alinhamento entre docentes, porém não há uma ferramenta oficial e padrão que permita concentrar, registrar, mensurar e compartilhar o desempenho em cada situação de aprendizagem e que permita a construção de um histórico de estratégias e a partilha com toda a equipe; fator que contribuiria para o desenvolvimento de competências, habilidades, valores e atitudes, com maior grau de eficácia, bem como um proveria maior embasamento para o planejamento de situações de aprendizagem com maior grau de assertividade para toda a equipe.

O presente projeto propõe o desenvolvimento de uma ferramenta colaborativa para auxiliar o docente na construção de seu plano de aula e eventual compartilhamento do histórico de suas ações pedagógicas com o intuito de fornecer subsídios para a construção de situações de aprendizagem mais assertivas.

2. 3 Fundamentação teórica

2.3.1 O Senac São Paulo

2.3.1.1 O Senac São Paulo e a educação profissional

A missão do Senac São Paulo é proporcionar o desenvolvimento de pessoas, por meio de ações educacionais que estimulem o exercício da cidadania e a atuação profissional transformadora e empreendedora, de forma a contribuir para o bem-estar da sociedade (Missão e Valores Institucionais, Senac São Paulo).

A educação para o mercado de trabalho tem como objetivo o desenvolvimento profissional e pessoal do indivíduo e o consequente impacto socioeconômico, positivo, na região em que ele está localizado. A instituição busca auxiliar na formação de profissionais que estejam atualizados e desenvolvam habilidades necessárias para o mercado contemporâneo; dentre as principais, apresentam-se, no quadro 1, os dez requisitos essenciais do profissional do futuro, segundo o Fórum Econômico Mundial.

Quadro 1 – Habilidades do profissional do futuro

10 habilidades do profissional do futuro
1. Flexibilidade cognitiva; 2. Negociação; 3. Orientação para servir; 4. Julgamento e tomada de decisões; 5. Inteligência emocional; 6. Coordenação com os outros; 7. Gestão de pessoas; 8. Criatividade; 9. Pensamento crítico; 10. Resolução de problemas.

Fonte: Adaptado de Whiting (2020)

O quadro 2, abaixo, demonstram os elementos que o Senac SP visa promover ao trabalhar com educação profissional:

Quadro 2 – Elementos promovidos por meio da educação profissional

Elementos a serem promovidos por meio da educação profissional
• o desenvolvimento de conhecimentos, saberes e competências profissionais; • uma formação que compreenda, além do domínio operacional de uma técnica ou prática de trabalho, a compreensão global do processo produtivo e de todos os conhecimentos que fundamentam a prática profissional; • o desenvolvimento da capacidade de aprendizagem permanente, fundamental para a sobrevivência num mundo do trabalho cada vez mais seletivo e que muda constantemente, bem como a capacidade de trabalhar em equipe e de buscar solução de problemas

Fonte: Adaptado de Senac (2016a)

2.3.1.2 A cultura do trabalho e a evolução da educação

Pelo pensamento, pela linguagem e pelo trabalho o homem dá sentido, conhece e modifica o mundo, entendido como o ambiente ou circunstância no qual vive, convive e transforma pela sua ação. (SENAC, 2005)

O trabalho permite o desenvolvimento do indivíduo e da coletividade. Ao longo do tempo o conceito de trabalho e relações trabalhistas ganharam novos contornos. A figura 1 a seguir demonstra as revoluções industriais e o impacto nas atividades executadas pelas empresas.

Figura 1 – Revoluções industriais

Fonte: Adaptado de SENAI-RS (2019)

Alguns trabalhos vão desaparecer, outros que nem sequer existem hoje se tornarão comuns. O que é certo é que a futura força de trabalho terá de alinhar o seu conjunto de habilidades para manter o ritmo. (SEBRAE, 2017)

Os profissionais também precisarão se adaptar, pois com fábricas ainda mais automatizadas novas demandas surgirão enquanto algumas deixarão de existir. Os trabalhos manuais e repetitivos já vêm sendo substituídos por mão de obra automatizada, e com indústria 4.0 isso tende a continuar. Por outro lado, as demandas em pesquisa e desenvolvimento oferecerão oportunidades para profissionais tecnicamente capacitados, com formação multidisciplinar para compreender e trabalhar com a variedade de tecnologia que compõe uma fábrica inteligente. (FERREIRA, 2018)

Observa-se, portanto, a eminente necessidade de constante aperfeiçoamento com o objetivo de desenvolver competências e habilidades para construir algo e não apenas executar. O quadro 3, apresentado abaixo, sintetiza as principais novidades das diretrizes curriculares nacionais para a educação técnica de nível médio e como a Proposta Pedagógica do Senac São Paulo trata este novo contexto:

Quadro 3 – Diretrizes Curriculares Nacionais e Proposta Pedagógica do Senac SP alinhados às necessidades contemporâneas.

Pontos principais das Diretrizes Curriculares Nacionais para a Educação Profissional Técnica de Nível Médio
• influência da evolução tecnológica nas relações de trabalho; • necessidade de formação de trabalhadores com mais propriedade do seu fazer e que avancem da realização de tarefas mecânicas para a gestão da própria ação; • novos desenhos e espaços para o trabalho, bem como outros formatos de relações trabalhistas (trabalhabilidade); Obs.: A Proposta Pedagógica também aborda o caldeirão de transformações pelas quais passa a sociedade contemporânea. São formatos inovadores de trabalho e emprego convivendo com modelos tradicionais, um leque de possibilidades que atendem os perfis e necessidades diversificadas, assim o novo torna-se natural

Fonte: Adaptado de Senac (2016a)

O quadro 4, a seguir, apresenta a evolução da educação ao longo do tempo.

Quadro 4 – Evolução da educação

Características	**Educação 1.0**	**Educação 2.0**	**Educação 3.0**	**Educação 4.0**
Período	Século XII	Século VIII e início do Século XIX	Final do Século XIX e meados do Século XX	Final do Século XX e início do Século XXI
Papel do professor	Ativo – centro do processo	Ativo – centro do processo	Organizador do processo	Organizador, orientador, mediador do processo
Papel do aluno	Passivo – mero receptor de conteúdos	Passivo – mero receptor de conteúdos	Ativo – responsável pela sua aprendizagem	Ativo – aprender a aprender e aprender a fazendo
Currículo	Integrado e baseado em conteúdos estáticos	Fragmentado e baseado em conteúdos estáticos	Integrados e atualizados constantemente.	Baseado em STEAM e metodologias ativas
Disseminação das informações	Fala dos mestres	Livros	Internet (web 1.0, 2.0, 3.0)	Internet mais potente e inteligente (web 4.0)
Objetivo	Promover o raciocínio, da linguagem e do pensar	Promover o treinamento, a memorização e o trabalho manual	Promover habilidades de acuidade mental	Desenvolver a autonomia, a capacidade de solucionar problemas nunca vistos.
Locais de estudo	Mosteiros com momentos mais ou menos flexíveis	Escolas, com momentos fixos.	Escolas com momentos flexíveis, com alternância do ensino presencial e a distância.	Ensino híbrido, sem distinção entre momentos presenciais e a distância
Gerações		Geração belle époque (1989 a 1992) e baby boomers (1945 a 1960)	Geração X (1960 a 1983), Y (1984 a 2000) e Z (início de 1990 a 2010)	Geração Alpha (a partir de 2010)

Fonte: Passos (2019)

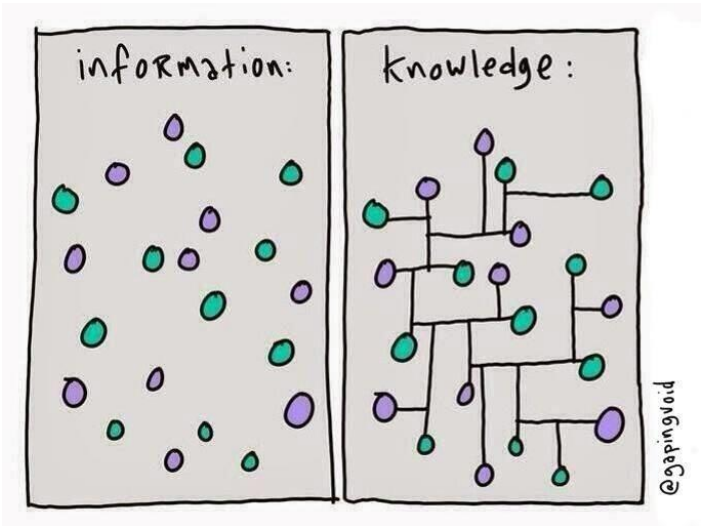
Segundo Goméz apud Führ (2018):

> A educação do século XXI encontra-se inserida no contexto de quarta revolução industrial que impacta a forma de pensar, de relacionar e de agir do ser humano. No percurso do tempo a educação sofreu uma acelerada metamorfose, pois o contexto social, econômico e político apresenta um novo cenário que requer outra postura do profissional inserido na era digital com as seguintes competências: capacidade de utilizar e comunicar de maneira disciplinada, crítica e criativa o conhecimento e as ferramentas simbólicas que a humanidade foi construindo através dos tempos; capacidade para viver e conviver democraticamente em grupos humanos cada vez mais heterogêneos, na sociedade global; capacidade de viver a atuar autonomamente e construir o próprio projeto de vida.

Observa-se, portanto, a necessidade de prover e correlacionar, cada vez mais, informações para que o indivíduo desenvolva conhecimento e possa tomar decisões com maior grau de embasamento ou encontrar novas formas para resolução de problemas existentes ou que atendam novas demandas.

De acordo com Davenport (1998), há uma relação direta entre conhecimento e informação. Podemos acumular milhares de informações ou fragmentos de várias fontes. No entanto, o conhecimento só é construído quando processamos a informação, quando a ela atribuímos sentido por meio da reflexão, da síntese, de sua articulação com repertórios pessoais e culturais. A figura 2, abaixo, apresentada por Escobar (2014), demonstra, visualmente, a diferença entre informação e conhecimento.

Figura 2 – Informação x conhecimento



Fonte: Escobar (2014)

Nesse contexto, Delors (2003) preconiza que a educação para o século XXI deve se apoiar em quatro pilares: conhecer, fazer, ser e conviver, conforme apresenta a figura 3 abaixo.

Figura 3 – Pilares contemporâneos da educação

Fonte: Senac (2016a)

2.3.1.3 O Jeito Senac de Educar

Educar é uma ação intencional e política. Possibilita ao indivíduo o desenvolvimento de competências, fundamentado em conhecimentos científicos e tecnológicos, aprendendo a conhecer, viver, conviver, agir e transformar sua vida e sua prática social, e a participar da sua comunidade. (Proposta Pedagógica, SENAC, 2005). O quadro 5 a seguir, contempla os pressupostos que regem a proposta educativa do Senac.

Quadro 5 - Pressupostos que regem a proposta educativa do Senac

Pressupostos que regem a proposta educativa do Senac
• Ampliar a visão crítica de mundo; • Participar da vida pública; • Defender seus direitos e ampliá-los; • Inserir-se e permanecer no mundo do trabalho, com desempenho de qualidade e com empreendedorismo; • Assumir responsabilidade social, com desempenho ético, de preservação do meio ambiente e de atenção à saúde individual e coletiva.

Fonte: Senac (2016a)

Os pressupostos, apresentados acima, tem como objetivo formar indivíduos dispostos a aprender a aprender sempre, autônomos e que acompanham às tendências do mercado de trabalho. A metodologia ativa de aprendizagem, presente em todos os cursos da instituição, coloca a aluno no centro do processo de capacitação e o faz interagir e refletir sobre problemas locais e globais.

Para atestar o desenvolvimento de competências, habilidades, atitudes e valores do indivíduo a instituição utiliza como principal instrumento a avaliação formativa, que segundo Cardinet *apud* Caseiro e Gebran (2008 ,p.3), pode ser definida como um método que:

> [...] visa orientar o aluno quanto ao trabalho escolar, procurando localizar as suas dificuldades para o ajudar a descobrir os processos que lhe permitirão progredir na sua aprendizagem. A avaliação formativa opõe-se à avaliação somativa que constitui um balanço parcial ou total de um conjunto de aprendizagens. A avaliação formativa se distingue ainda da avaliação de diagnóstico por uma conotação menos patológica, não considerando o aluno como um caso a tratar, considera os erros como normais e característicos de um determinado nível de desenvolvimento na aprendizagem.

A instituição estudada, adota múltiplos critérios de observação e análise ao aplicar o instrumento mencionado acima: visão empreendedora, projeções de futuro, relacionamento interpessoal, leitura de cenários, domínio técnico científico entre outros. Estes parâmetros são fundamentais para a elaboração de feedbacks individuais e constantes ao aluno que contribuem para sua evolução ao longo do curso, bem como norteiam o corpo docente para o aperfeiçoamento ou revisão de suas ações pedagógicas.

Diante do cenário apresentado, o docente do Senac São Paulo atua como mediador que contribui com a construção ativa do conhecimento dos alunos, onde a ênfase do processo de ensino aprendizagem não está no conteúdo, mas sim no desenvolvimento de competências em função da mobilização de conhecimentos, habilidades e atitudes.

O erro do aluno é um caminho para que o docente compreenda o que o aluno já sabe e o que não sabe, podendo assim encontrar caminhos para intervir no processo de ensino e aprendizagem. (Senac, 2016a)

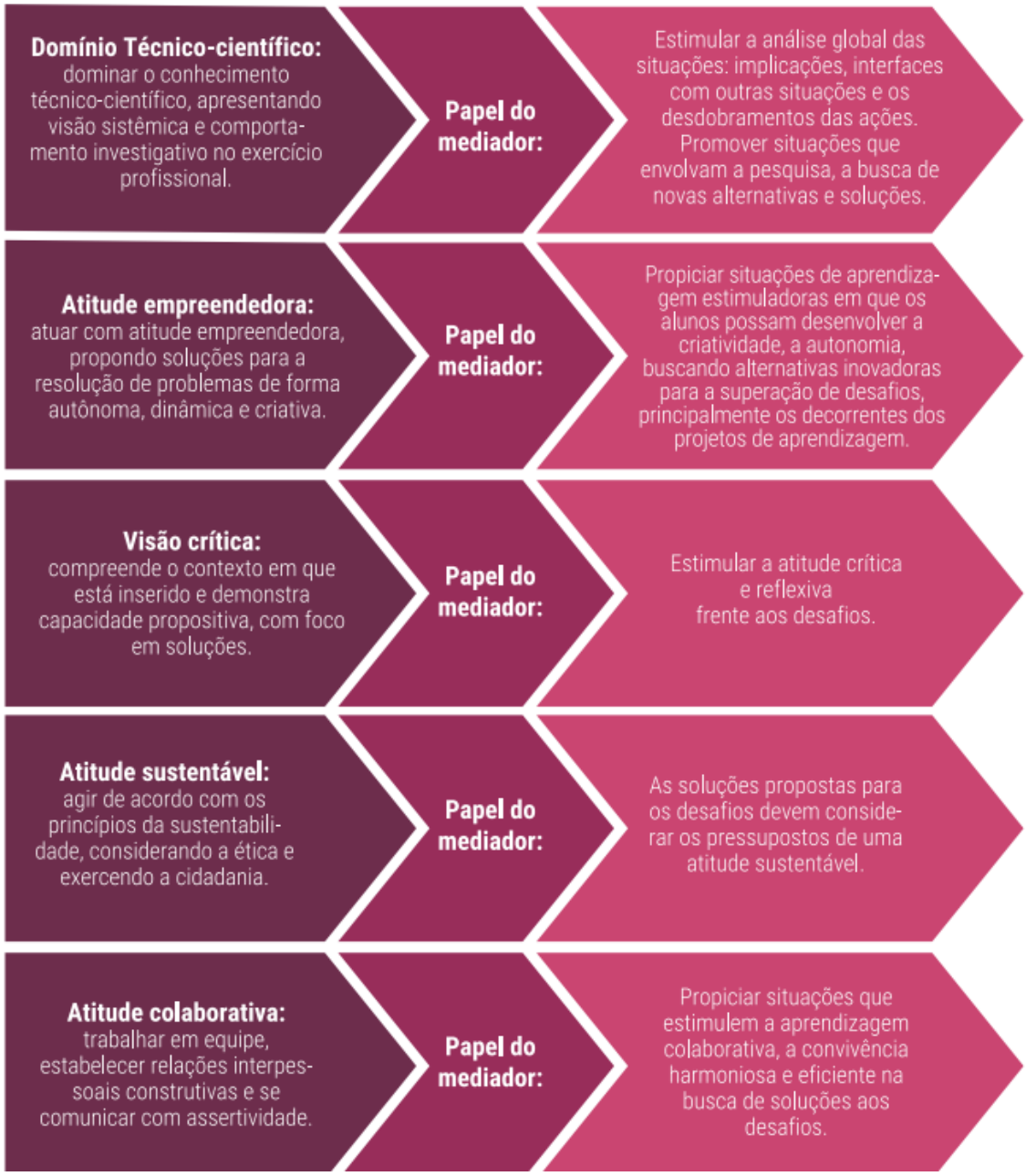
Os planos de cursos, também contam com elementos denominados marcas formativas, cujo objetivo é promover a formação de um profissional diferencial para o mercado de trabalho, com o selo da instituição.

Segundo Senac (2016a, p. 27):

> As marcas formativas apontam para o que queremos promover na aprendizagem dos alunos, as formas como deve atuar, o que é fundamental dominar e desenvolver, visando a formação de um profissional diferenciado.

A figura 4, a seguir, apresenta as marcas formativas e a atuação do docente como mediador do processo de ensino aprendizagem.

Figura 4 – Marcas formativas Senac



Fonte: Senac (2016a)

2.3.1.4 Modelo de desenvolvimento de competências

O modelo curricular do Senac é estruturado com base no desenvolvimento de competências. Uma competência, por sua vez, é definida como "ação/fazer profissional observável, potencialmente criativo, que articula conhecimentos, habilidades, atitudes/valores e permite o desenvolvimento contínuo". (SENAC, 2015, p. 19)

O quadro 6, a seguir, apresenta os principais pareceres do Conselho Nacional de Educação acerca do conceito de competência aplicada Educação Profissional de Nível Técnico.

Quadro 6 - Principais pareceres do Conselho Nacional de Educação

Parecer	**Definição de competência**
CNE/CEB nº 16/1999: trata das Diretrizes Curriculares Nacionais para a Educação Profissional de Nível Técnico e regulamenta a Lei de Diretrizes e Bases da Educação	capacidade de mobilizar, articular e colocar em ação valores, conhecimentos e habilidades necessárias ao desempenho eficiente e eficaz de atividades requeridas pela natureza do trabalho.
CNE/CEB nº 11/2012: trata da necessidade dos currículos promoverem o desenvolvimento de competências (profissionais), a educação para a vida.	A educação para vida poderá propiciar aos trabalhadores o desenvolvimento de conhecimentos, saberes e competências que os habilitem efetivamente para analisar, questionar e entender os fatos do dia a dia com mais propriedade, dotando-os, também, de capacidade investigativa diante da vida, de forma mais criativa e crítica, tornando-os mais aptos para identificar necessidades e oportunidades de melhorias para si, suas famílias e a sociedade na qual vivem e atuam como cidadãos.

Fonte: Adaptado Senac (2016a)

As competências são constituídas por meio dos elementos presentes que são mobilizados em situações de aprendizagem: conhecimentos, atitudes e valores, habilidades. O quadro 7, a seguir, apresenta uma breve descrição dos insumos necessários para promover a formação de competência.

Quadro 7 - Principais pareceres do Conselho Nacional de Educação

Insumos para formar competências	
Conhecimentos	Base do saber. É tudo aquilo que o aluno precisa conhecer para desempenhar a prática descrita na competência. É composto de conceitos, contextos históricos e princípios técnico-científicos e legais que fundamentam a prática profissional.
Atitudes e valores	Relacionam-se com as percepções e concepções a respeito do mundo, do necessário comprometimento relacional e social dos profissionais com o próprio trabalho. As atitudes e os valores qualificam o fazer profissional, pois orientam a postura que se exige de um profissional ético e cidadão.
Habilidades	Referem-se às capacidades necessárias para que o aluno saiba executar determinado fazer profissional, por isso vai muito além de atividades a serem realizadas ou do desempenho de uma simples tarefa.

Fonte: Adaptado de Senac (2016a)

Em relação as características de uma competência, contatam-se algumas peculiaridades:

- É observável: possuem uma ação, ou conjunto de ações que podem ser facilmente observados em situações de aprendizagem e na atividade profissional;
- Estimula a criatividade: não possui um roteiro de ações pré-estabelecidas, propicia e incentiva o encontro de novas possibilidades para resolução de problemas;
- Proporciona o desenvolvimento contínuo: exige o aperfeiçoamento constante dos saberes (conhecimentos, habilidades e atitudes) e, consequentemente, acarreta em atualizações frequentes sobre a percepção do mundo.

O ato de evidenciar o desenvolvimento de competências é estabelecido por meio dos indicadores presentes no plano de curso. Eles têm como papel principal monitorar e acompanhar o processo formativo, mostrando a performance obtida pelo aluno em relação as expectativas de aprendizagem. O quadro 8, a seguir, as características dos indicadores de competências.

Quadro 8 - Características dos indicadores de competências

Características dos indicadores de competências
• descreve uma ação observável, ou seja, uma ação realizada pelo aluno que evidencia o seu desenvolvimento em relação ao desenvolvimento da competência; • relaciona-se a um ou mais elementos da competência (Conhecimento/Atitudes/Habilidades); • é compreensível para todos os envolvidos no processo: alunos e agentes educacionais; • define o parâmetro pelo qual o aluno será avaliado, precisa ser compreensível e compartilhado com os alunos.

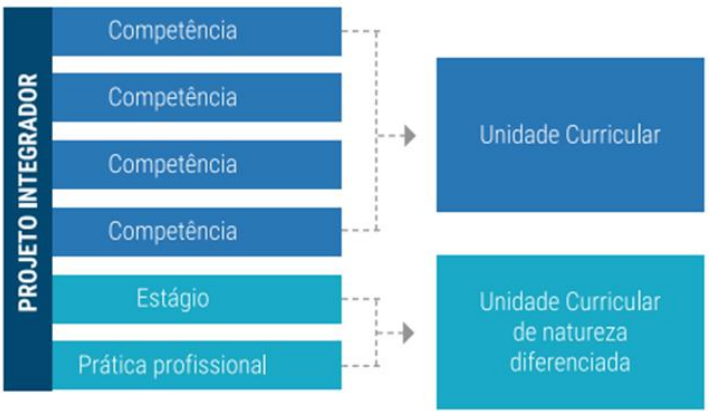
Fonte: Adaptado de Senac (2016a)

Para a avaliação do desempenho dos alunos, o docente deve guiar-se pelos indicadores de competência. É importante saber que a avaliação no Senac pretende ir além da simples atribuição de valor a determinada atividade realizada pelo aluno, como costuma ser bastante comum no ensino tradicional. A avaliação deve ser considerada indicativo do avanço do aluno rumo à excelência na execução de seu saber profissional. (SENAC, 2015)

2.3.1.5 Organização curricular

A configuração curricular dos cursos ofertados pelo Senac, conforme apresenta a figura 5 abaixo, está voltada para o desenvolvimento de competências. No modelo vigente a Unidade Curricular (UC) é constituída de uma ou mais competências.

Figura 5 – Organização curricular



Fonte: Senac(2016a)

As competências, por sua vez, podem ser desdobradas em:

- Elementos: insumos que deverão ser mobilizados para o desenvolvimento de competência;
- Indicadores: evidenciam o desenvolvimento da competência no desempenho do aluno.

O quadro 9, abaixo, apresenta um exemplo prático da relação entre competência, elementos e indicadores.

Quadro 9 – Relação entre competência, elementos e indicadores.

Competência Apoiar as atividades de compra de equipamentos, materiais, produtos e serviços.	**Elementos**	• Tipos de documentos: pedido de compra, nota fiscal, romaneio de carga, ficha de inspeção, ficha de cautela, requisição de materiais. Métodos de ressuprimento: contínuo, periódico e parâmetros de estoques. • Noções de planejamento de compras: características de consumo de materiais, previsão de demanda e levantamento das necessidades de compra – Lote Econômico de Compras (LEC). • Preencher relatórios e documentos. • Controlar estoques. • Sigilo no tratamento das informações. • Flexibilidade no relacionamento com fornecedores.
	Indicadores	• Providencia a reposição de produtos e materiais, conforme demanda, acompanhamento e pedido da área de suprimentos e demais área da organização. • Apoia pesquisa, seleção e cadastro de produtos e materiais, de acordo com os princípios de relacionamento com fornecedores. • Controla o fluxo de entrega dos fornecedores, acompanhando o processo dos pedidos e conferência de mercadorias, de acordo com as características dos processos de suprimento da empresa. • Identifica a necessidade de compra (o quê, quando e quanto), de acordo com o método de ressuprimento adequado à política de estoques da empresa.

Fonte: Senac(2016a)

2.3.1.6 Plano coletivo trabalho docente e plano de aula

O planejamento da prática educativa no Senac é materializado nos Planos Coletivos de Trabalho Docente (PCTD) e nos Planos de Aula (SENAC, 2016b).

Na visão do Senac, o Plano Coletivo de Trabalho Docente:

> é um plano elaborado por docentes e área técnica com a função de alinhar os objetivos e as estratégias que serão desenvolvidas durante o curso, com destaque para o projeto integrador. Esse documento favorece a coerência e a coesão das ações do curso todo, além de permitir o encadeamento das competências, um exercício de interdisciplinar idade para integralidade da formação (SENAC, 2016b, p. 13).

O PCTD funciona como um norteador das situações de aprendizagem, com objetivo de desenvolver competências (SENAC, 2016b). Sua elaboração, como próprio nome diz, é coletiva, realizada por docentes e acompanhada pelas equipes técnicas.

A construção do PCTD envolve os seguintes passos (SENAC, 2016b, p. 22-27), sempre antes do início de cada turma:

- Contextualização, quando os docentes conhecem os materiais produzidos anteriormente sobre curso (O Plano de Curso e o Plano de Orientação para Oferta) e o perfil dos alunos. Além disso, é o momento de interação entre os docentes e compartilhamento e alinhamento de concepções e metodologias;
- Definição do tema gerador, que estabelecerá os objetivos do projeto integrador e promoverá a articulação entre as competências que se quer desenvolver no curso;
- Insumos das UCs-Competência para o desenvolvimento do Projeto Integrador, momento em que as diversas unidades curriculares do curso são articuladas com as competências necessárias para o desenvolvimento do projeto integrador;
- Organização do processo de ensino e aprendizagem, em que é definida a articulação e diálogo entre as diversas unidades curricular ares, bem como a definição de possibilidades de abordagens pedagógicas, processos de avaliação e feedback aos alunos;

O documento é por fim sistematizado e registrado, sem que isso signifique, no entanto, que ele não esteja aberto a replanejamentos e alterações posteriores.

Após o trabalho coletivo de elaboração do PCTD e do estabelecimento das responsabilidades, cada docente elaborará o seu Plano de Aula, que é o “documento que sistematiza o planejamento da Unidade Curricular realizado pelo docente responsável por ela,

que tem o propósito de organizar o processo de ensino e aprendizagem, sempre em função da aprendizagem do aluno" (SENAC, 2016b, p. 28).

O plano de aula é o detalhamento das situações de aprendizagem (SENAC, 2016b). Nesse sentido, GIL (2020) afirma que:

> O que o professor tem a fazer, nesse caso, é principalmente especificar os conteúdos, cuidando para que cada um de seus tópicos seja desenvolvido mediante a utilização das estratégias e dos recursos mais adequados, com rigorosa previsão do tempo e das atividades que ficarão a cargo dos alunos (GIL, 2020, p. 31).

Do ponto de vista da estrutura, "São elementos conceituais do plano de aula: estrutura didática; temática; objetivo; conteúdo programático; estratégias e recursos didáticos; duração e referências." (TAKAHASHI; FERNANDES, 2004, p. 115). O Senac orienta (SENAC, 2016b, p. 30-33) que esses elementos sejam informados no plano de aula pelo docente através da seguinte lista de questionamentos:

- Para quem? (O perfil do aluno);
- Por quê? (Identificação da competência a ser desenvolvida pela situação de aprendizagem);
- O que? (identificação dos saberes a serem mobilizados na situação de aprendizagem -preferencialmente, deve envolver as três dimensões da competência: conhecimentos, habilidades e atitudes);
- Como faremos? (A elaboração de situações de aprendizagem que permitam que o aluno do perfil identificado possa desenvolver a competência determinada como objetivo, mobilizando os conhecimentos, habilidades e atitudes vinculadas a esta competência).

Finalmente, análise posterior da dinâmica da aula, aferindo o atingimento das expectativas tanto dos alunos quanto dos docentes, servirá como subsídio para um eventual replanejamento do plano de aula (SENAC, 2016b).

Deve-se ressaltar aqui que, diferentemente do PCTD, o plano de aula é de elaboração individual do docente. Literatura recente, no entanto, vem demonstrando as vantagens do planejamento colaborativo mesmo no nível do Plano de Aula:

> O planejamento colaborativo entre professores permite que os planos de aula sejam elaborados de forma criativa e eficiente para garantir que o aluno desenvolva as habilidades esperadas e atribui funções sociais para os conteúdos de sala de aula. (SILVA; VIANA, 2021, sem informação de página).

Um bom Plano de Aula precisa ser coeso e coerente e garantir a continuidade e complementaridade entre as situações de aprendizagem, visando o objetivo final que é o alcance da competência traçada como meta. (SENAC, 2016b, p. 31).

Andrade (2015, p.8) define situação de aprendizagem como:

> ... estratégias utilizadas para melhorar a aprendizagem do aluno na mediação dos conhecimentos mediados também pelas atividades práticas. Elas são aplicáveis a qualquer segmento de ensino desde a Educação Infantil ao Ensino Superior.

A situação de aprendizagem, por sua vez, deve englobar os itens que compõem a competência (conhecimento, habilidades e atitudes), com o objetivo de contribuir para a formação do indivíduo. Por fim serão elencados os indicadores de competência que podem ser atrelados a situação de aprendizagem, a figura 6 a seguir, exemplifica a relação entre os atores que constituem a situação de aprendizagem.

Figura 6 – Relação entre os atores que constituem a situação de aprendizagem

Fonte: Senac(2016b)

O plano de aula, portanto, é o documento desenvolvido pelo docente que sintetiza as situações de aprendizagem que serão utilizadas para o desenvolvimento de competências. O quadro 10, abaixo, demonstra um modelo documento proposto pelos manuais da instituição.

Quadro 10 – Relação entre competência, elementos e indicadores.

Item do Plano de Aula	**Descrição / Exemplo**
Competência	A Competência é o ponto de partida para a elaboração das situações de aprendizagem. Transcrever a competência do Plano de Curso. **Exemplo:** UC5 - Preparar e apresentar produções da culinária
Tema gerador do Projeto Integrador	Indicar o tema gerador que foi discutido e definido no plano coletivo de trabalho docente. O tema gerador precisa ser significativo para os alunos, ser desafiador, estimular a pesquisa e investigação, estar contextualizado com a realidade local e mobilizar as competências. **Exemplo:** Elaboração de produções culinárias em eventos.
Título da situação de aprendizagem	Inserir título da situação de aprendizagem (lembrar que a situação de aprendizagem é um conjunto de atividades articuladas e complementares que visam o desenvolvimento de uma ou mais competências ou a construção de um determinado saber). Para sua elaboração, considere o objetivo de aprendizagem para o aluno, nomeando o agrupamento de atividades programadas para esse objetivo. A situação de aprendizagem deve articular os elementos da competência e os indicadores relacionados que se pretende observar. Deverão ser previstas tantas situações de aprendizagem quanto forem necessárias para o desenvolvimento da competência **Exemplo:** Preparo de pratos da Cozinha Regional Brasileira
Duração da situação de aprendizagem	Indicar a duração prevista, o número de horas necessárias para a realização de uma situação de aprendizagem, que tem duração variável e não está ligada necessariamente ao tempo da hora aula, podendo ter quantas horas o docente julgar necessárias. **Exemplo:** 12 horas.
Descrição das atividades que compõem a situação de aprendizagem	Descrever as atividades que compõem a situação de aprendizagem, indicando as estratégias que serão utilizadas no desenvolvimento de determinada situação de aprendizagem. Essas atividades são complementares e encadeadas. É importante diversificar as atividades de forma a atender as necessidades de aprendizagem de todos os alunos. **Exemplo:** Atividade 1 – Discussão sobre os principais pratos que os alunos apreciam da Cozinha Regional Brasileira e listá-los. Atividade 2 – Apresentação de vídeo sobre Turismo Gastronômico no Nordeste Brasileiro. Atividade 3 – Identificação dos pratos listados pelos alunos e a culinária regional, observando: características, ingredientes, formas de preparo.
Descrição das contribuições da Situação de Aprendizagem para o Projeto Integrador	Descrever quais serão os insumos que subsidiarão a construção do projeto integrador, gerados no decorrer do desenvolvimento da competência ou das situações de aprendizagem ou das atividades. Esses insumos podem ser pesquisas, produções, relatórios e outros documentos ou aprendizagens que contribuam com a construção do projeto. **Exemplo:** As aprendizagens sobre Cozinha Regional Brasileira e Internacional trarão desafios de contextualização para o tema do projeto dos alunos que é a culinária vegetariana, resultando em pesquisas de alimentos e cardápios.
Elementos	Liste os elementos relacionados a esta situação de aprendizagem. Lembre-se de considerar elementos de todas as naturezas: conhecimento, habilidade, valor/atitude.
Indicadores	Descreva os indicadores da competência que se relacionam e que podem ser observados em determinada situação de aprendizagem. **Exemplo:** Seleciona ingredientes brasileiros, de acordo com as especificidades de cada produção culinária
Avaliação	Indicar como o aluno será avaliado em determinada situação de aprendizagem, incluindo o instrumento utilizado para isso, tendo como foco a avaliação contínua. **Exemplo:** Observação da construção coletiva da lista de principais pratos da cozinha nordestina e recebimento e análise do fichamento sobre a pesquisa sobre pratos listados pelos alunos.
Recursos	Defina os recursos a serem utilizados no desenvolvimento de determinada situação de aprendizagem. **Exemplo:** Projetor, computador

Fonte: Senac(2016b)

2.3.2 Internet e World Wide Web

2.3.2.1 Internet

A internet é uma rede de computadores - e, cada vez mais, não apenas de computadores - interconectados. Tanto os computadores, como TVs, celulares, consoles de jogos, veículos de transporte, equipamentos de monitoramento e segurança etc. conectados a esta rede são chamados de hospedeiros ou sistemas finais. A conexão entre sistemas finais pelos enlaces de comunicação (links - meios físicos, como cabos ou ondas de rádio, que transmitem os pacotes de informação de um ponto ao outro), e por computadores de pacotes (roteadores e *switches*, que funcionam como entroncamentos dos enlaces). Comutadores de pacotes e enlaces de comunicação são organizados em redes, chamadas de provedores de serviços de internet (ISP), e é através destes provedores que os sistemas finais acessam a internet. (KUROSE, ROSS, 2011).

É nesse momento que começa ficar mais claro que a internet é, na verdade, uma "rede de redes" (KUROSE, ROSS, 2011, p. 23). Se um sistema final usando os serviços de um determinado ISP necessita se comunicar com um sistema final que utiliza os serviços de outro a ISP (um computador doméstico acessando os conteúdos de um site, por exemplo), os dois ISPs usam os serviços de um ISP de nível mais alto (capaz de uma comunicação mais veloz) para se comunicar. (KUROSE, ROSS, 2011).

Para garantir o envio e a integridade dos dados trocados entre os diversos sistemas finais através dos múltiplos enlaces e comutadores, foram estabelecidos diversos protocolos de comunicação que regulam o tráfego de informações da internet. Os dois protocolos mais importantes são o TCP (T*ransmission Control Protocol*) e o IP (*Internet Protocol*). Ambos dão nome a arquitetura que é padrão da internet atual, chamado protocolo TCP/IP.

Arquitetura do protocolo TCP/IP está organizado em quatro camadas, cada uma delas organizadas por protocolos específicos: a camada física, de rede, de transporte e de aplicação.

O quadro 11, abaixo, adaptada de Alves (2019), apresenta os principais protocolos de rede utilizados atualmente.

Quadro 11 – Principais protocolos de rede utilizados atualmente

Protocolo	**Descrição**
HTTP (*HyperText Transfer Protocol*)	Utilizado pela WWW, permite o acesso a documentos de hipertexto em formato HTML. Esse protocolo torna possível a interligação de diversos documentos por meio de hipertextos.
FTP (*File Transfer Protocol*)	Utilizado na transferência de arquivos pela rede.
SMTP (*Simple Mail Transfer Protocol*)	Utilizado na transferência de mensagens eletrônicas do servidor de correio do remetente para o servidor de correio do destinatário.
POP (*Post Office Protocol*)	Permite a recuperação das mensagens armazenadas em um servidor de correio eletrônico.
TELNET (*Teletype Network*)	Protocolo que permite efetuar login remoto em máquinas nas quais o usuário possui uma conta de acesso. É possível ainda executar remotamente aplicações por meio de um terminal
ICMP (*Internet Control Message Protocol*)	Serve para monitoramento contínuo da rede pelos roteadores ou por programas de gerenciamento.
IGMP (*Internet Group Management Protocol*)	Permite o endereçamento de um datagrama IP a um conjunto de dispositivos que fazem parte de um grupo representado por endereço IP da classe D.
ARP (*Address Resolution Protocol*)	Possibilita encontrar o endereço de um adaptador de rede Ethernet que corresponde a um endereço IP.
RARP (*Reverse Address Resolution Protocol*)	Permite encontrar o endereço IP correspondente ao endereço físico de um adaptador de rede *Ethernet*, utilizando seu endereço MAC.
UDP (*User Datagram Protocol)*	Utilizado por uma aplicação na transferência de dados, denominados pacotes, sem preocupação de estabelecimento de uma conexão prévia ou na confiabilidade dos dados.

Fonte: Adaptado de Alves (2019)

2.3.2.2 *World Wide Web*

A internet também pode ser descrita (KUROSE, ROSS, 2011) com uma infraestrutura de provimento de serviços para aplicações distribuídas. Aplicações distribuídas são aplicações que envolvem a troca de informações entre diversos sistemas finais. Serviços de e-mail, redes sociais, jogos distribuídos, compartilhamento de arquivos e *streaming* são exemplos de aplicações distribuídas que usam a infraestrutura da internet. A *Web* também deve ser incluída nesta lista.

Ljubomir (2016, p. 398) define: "A *World Wide Web* (WWW ou, simplesmente, a *Web*) é um sistema distribuído de documentos ligados por hyperlinks e hospedados em servidores Web pela Internet".

Como qualquer aplicação distribuída, a WWW permite a comunicação entre um computador que hospeda um determinado recurso (páginas web, documentos, multimídia etc,), chamado de servidor, E um ou mais sistemas finais que solicitam esses recursos do servidor - que são os clientes. Para atuar como servidor, um sistema final executa um programa chamado o servidor *web*, e para atuar como cliente, o sistema final executa um programa chamado cliente *web* (Os navegadores de internet são exemplos de programas cliente web).

Cada recurso na web pode ser acessado através de um identificador exclusivo denominado URL (*Uniform Resource Locator). "*O URL não apenas identifica exclusivamente um recurso, mas também especifica como acessá-lo, assim como o endereço de uma pessoa pode ser usado para acessá-la" (LJUBOMIR, 2016, p. 400). A URL é uma cadeia de caracteres (string) como, por exemplo http://www.w3.org/Consortium/mission.html, e cada parte dela tem uma função, como descrito abaixo:

- http é o esquema, especificando o protocolo a ser usado para acessar o recurso;
- www.w3.org é o *host* ou hospedeiro, e especifica o servidor que hospeda o documento. Seu nome é traduzido para o endereço definido pelo protocolo IP através do Serviço de Nomes de Domínio (DNS) (ALVES, 2019);
- Consortium/mission.html é o *path* ou caminho, a localização do recurso (neste caso, uma página html) dentro diretório raiz do servidor *Web*.

A definição da *World Wide Web* Como aplicação distribuída, e a importância dos processos cliente e servidor no seu funcionamento nos levam a necessidade de detalhar arquitetura cliente-servidor.

2.3.2.3 Arquitetura Cliente-Servidor

Arquiteturas de *software* integram os vários componentes que tornam possível uma aplicação (o código da própria aplicação, o banco de dados, o versionamento etc.). Os modelos ou padrões de arquitetura são uma maneira de apresentar, compartilhar e reutilizar conhecimento sobre sistemas (SOMMERVILLE, 2019). O padrão de arquitetura é "… como uma descrição estilizada, abstrata, das práticas recomendadas que foram testadas e aprovadas em diferentes subsistemas e ambientes" (SOMMERVILLE, 2019, p. 155).

Um dos diversos tipos de arquitetura é a cliente-servidor. Se organizado dentro de um padrão cliente-servidor, um sistema é um conjunto de serviços e servidores associados, e de clientes que acessam esses serviços.

Ainda segundo Somerville (2019), os três principais componentes da arquitetura cliente servidor são os servidores (componentes de software que fornecem os serviços), clientes (programas que executam as demandas de serviço oferecidas pelos servidores) e uma rede que permite que o acesso dos clientes aos serviços. Essa comunicação é feita através da utilização de protocolos de requisição-respostas, como HTTP.

Uma das principais vantagens da arquitetura cliente servidor é sua possibilidade de utilização em sistemas distribuídos que são acessados por meio da internet. Nesseа caso o,

> usuário interage com o programa que está sendo executado em seu computador local, como um navegador web um aplicativo em um dispositivo móvel. Estes interagem com o programa que está sendo executado em um computador remoto como um servidor web. O computador remoto fornece serviços, como acesso a páginas web que estão disponíveis para clientes externos (SOMMERVILLE, 2019, p. 470).

Cada cliente interage apenas com os servidores, e não com os outros clientes[1].

Como definem Tanembaum e Van Steen, "um sistema distribuído é um conjunto de computadores independentes que se apresenta seus usuários como um sistema único e coerente" (TANENBAUM, VAN STEEN, 2007, p. 1).

Em geral, cada instância do processo servidor executada em uma máquina diferente, e um software de balanceamento de carga atua para que o mesmo volume de trabalho seja distribuído para todos eles, aumentando o volume de transações sem degradação da resposta para os clientes.

Para que isso funcione, o projeto de um sistema cliente servidor distribuído deve atentar-se à a distinção entre os processos de apresentação, criação e processamento das informações, bem como das interfaces entre eles. A estruturação dessas distinções efetivada através de várias camadas lógicas:

- camada de apresentação, que gerencia a interação com usuário;
- camada de manipulação de dados, que gerencia os dados passados para e recebidos do cliente;
- camada de processamento da aplicação, que gerencia a lógica de fornecimento das funcionalidades;
- camada de banco de dados, que armazena os dados e gerencia serviços e transações de consulta destes dados.

Essas quatro camadas são organizadas de maneiras diferentes em diferentes arquiteturas cliente servidor.

Na arquitetura cliente servidor de duas camadas, o modelo *thin-client* (ou cliente magro) implementa a camada de apresentação no cliente e as demais camadas no servidor; enquanto o modelo *fat-client* (ou cliente gordo) as camadas de apresentação e processamento da aplicação

[1] Embora o modelo cliente servidor não se restringe apenas a esse tipo de uso: "também é possível utilizá-lo como um modelo de interação lógica em que o cliente e o servidor rodam no mesmo computador" (SOMMERVILLE, 2019, p. 470)

são implementadas no cliente, e as camadas de banco de dados e gerenciamento de dados no servidor.

Para Sommerville, a vantagem do *thin-client* é a simplicidade: "se o navegador web for utilizado como cliente, não há necessidade de instalar qualquer software" (SOMMERVILLE, 2019, p. 474); a desvantagem é que toda a carga de processamento fica do lado do servidor, trazendo problemas de escalabilidade e desempenho. Além disso, esse modelo exige mais tráfego de rede. Os modelos *fat-client* minimizam o problema de sobrecarga do processador e da rede, mas demanda a instalação de software do cliente, e reinstalações quando necessário, o que aumenta a complexidade de gerenciamento e o custo. O aumento da capacidade de processamento das máquinas cliente diminuiu a fronteira entre *thin-client* e *fat-client*, e "poucas são as aplicações que implementou todo o processamento no servidor remoto" (SOMMERVILLE, 2019, p. 476).

Para sanar os problemas de desempenho típicos do cliente magro, e de gerenciamento típicos do gerente gordo, a solução encontrada foi uma arquitetura entre as diversas camadas são executados por processadores diferentes -as chamadas arquitetura cliente seu servidor multicamadas. Esse tipo de arquitetura permite maior escala habilidade, uma vez que "as camadas no sistema podem ser gerenciadas de maneira independente, com outros servidores adicionados à medida que a carga aumenta" (SOMMERVILLE, 2019, p. 477).

Não elaboração de um projeto, a escolha entre as arquiteturas cliente servidor no modelo cliente magro, cliente gordo, ou multicamadas depende de uma série de fatores que podem ser sintetizados na tabela 1, abaixo, extraída de Sommerville (2019):

Tabela 1 – Uso de padrões de arquitetura cliente-servidor

Arquitetura	Aplicações
Arquitetura cliente-servidor de duas camadas com clientes magros	Aplicações de sistemas legados que são utilizadas quando é impraticável separar o processamento da aplicação e a manipulação dos dados. Os clientes podem acessar esses serviços, conforme será discutido na Seção 17.4. Aplicações computacionalmente intensivas, como compiladores com poucos ou nenhum requisito de manipulação de dados. Aplicações intensivas em termos de dados (navegação e consulta) com processamento da aplicação não intensivo. Navegação simples na internet é o exemplo mais comum de uma situação em que essa arquitetura é utilizada.
Arquitetura cliente-servidor de duas camadas com clientes gordos	Aplicações nas quais o processamento é fornecido por software de prateleira (por exemplo, Microsoft Excel) no cliente. Aplicações nas quais é necessário o processamento computacionalmente intensivo de dados (por exemplo, visualização de dados). Aplicações móveis nas quais a conectividade via internet não pode ser assegurada. Portanto, o processamento local usando informações em cache do banco de dados é possível.
Arquitetura cliente-servidor multicamadas	Aplicações de larga escala com centenas ou milhares de clientes. Aplicações nas quais tanto os dados quanto a aplicação são voláteis. Aplicações nas quais dados de várias fontes são integrados.

Fonte: SOMMERVILLE (2019, p. 477)

2.3.2.4 Padrões MVC e MTV

Outro padrão muito popular no desenvolvimento de aplicações web, e que também faz uso da estrutura em camadas é o padrão MVC (*Model-View-Controller*), que tem "como objetivo isolar ao máximo a camada de apresentação de um sistema" (ZENKER, SANTOS, COUTO, 2019, p. 100). A vantagem da organização da interface do usuário seguindo as regras deste padrão é que ele

> permite que os dados sejam alterados independentemente de sua representação e vice-versa. Apoia a apresentação dos mesmos dados de maneiras diferentes, exibindo as alterações feitas em uma representação em todas as demais" (SOMMMERVILLE, 2019, p. 155).

As camadas no padrão MVC são o modelo (*model*), que contém que contém os dados e comportamentos da aplicação, e grosso modo, é equivalente as camadas de banco de dados e processamento da aplicação; visão (*view*), que é a camada de apresentação ao usuário; e o controlador (*controller)*, o que faz a interligação entre a camada de interface do usuário e a camada de dados e comportamentos.

Segundo Zenker, Santos e Couto (2019), o modelo representa as identidades e operações presentes do banco de dados, a *view* representa os componentes visuais e a interface a ser mostrada usuário e o controlador implementa regra de negócio da aplicação, interpretando e respondendo as requisições feitas pelo usuário

O modelo MTV (*model-template-view*) é uma variação do MVC, E muito utilizado em freme Work para desenvolvimento de aplicações como, o Django. No MTV, model é equivalente ao modelo do MVC; tem Plate é a lógica de apresentação ao usuário, ou seja, a view do MVC; E, finalmente, a view do MTV implementa as regras de negócio, sendo equivalente ao controlador do MVC (Maciel, 2020).

Discutiremos com mais detalhes o modelo MTV no item **2.3.6.2.**

2.3.3 Banco de dados e SGBD

2.3.3.1 Banco de dados

A evolução dos bancos de dados foi e continua sendo de constante melhoria, trazendo mais segurança e rapidez para as informações ou dados nele salvas. Para Korth, Silberschatz e Sudarshan (2012), um banco de dados "[...] é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico".

A ideia de banco de dados relacionais surgiu nas décadas de 1960 e 1970 na IBM através de pesquisas de funções de automação de escritório, nesse período foi descoberto que era custoso empregar muitas pessoas para armazenar e indexar arquivos.

Muitas pesquisas foram conduzidas durante este período, cujos modelo hierárquicos, de rede e relacionais e outros modelos foram descobertos, bem como muita tecnologia utilizada hoje em dia.

Em 1970 um pesquisador da IBM - Ted Codd - publicou o primeiro artigo sobre bancos de dados relacionais. Este artigo tratava sobre o uso de cálculo e álgebra relacional para permitir que usuários não técnicos armazenassem e recuperassem grande quantidade de informações.

Codd visionava um sistema onde o usuário seria capaz de acessar as informações através de comandos em inglês, onde as informações estariam armazenadas em tabelas.

Devido à natureza técnica deste artigo e a relativa complicação matemática, o significado e proposição do artigo não foram prontamente realizados. Entretanto ele levou a IBM a montar um grupo de pesquisa conhecido como *System* R (Sistema R).

O projeto do Sistema R era criar um sistema de banco de dados relacional o qual eventualmente se tornaria um produto. Os primeiros protótipos foram utilizados por muitas organizações, tais como MIT *Sloan School of Management* (uma escola renomada de negócios norte-americana). Novas versões foram testadas com empresas aviação para rastreamento do estoque.

Eventualmente o Sistema R evoluiu para SQL/DS, o qual posteriormente tornou-se o DB2. A linguagem criada pelo grupo do Sistema R foi a *Structured Query Language* (SQL) - Linguagem de Consulta Estruturada. Esta linguagem tornou-se um padrão na indústria para bancos de dados relacionais e hoje em dia é um padrão ISO (*International Organization for Standardization*). A ISO é a Organização Internacional de Padronização.

2.3.3.2 Os primeiros do mercado

Mesmo a IBM sendo a companhia que inventou o conceito original e o padrão SQL, eles não produziram o primeiro sistema comercial de banco de dados. O feito foi realizado pela *Honeywell Information Systems Inc*., cujo sistema foi lançado em junho de 1976. O sistema era baseado em muitos princípios do sistema que a IBM concebeu, mas foi modelado e implementado fora da IBM.

O primeiro sistema de banco de dados construído baseado nos padrões SQL começaram a aparecer no início dos anos 80 com a empresa *Oracle* através do *Oracle* 2 e depois com a IBM através do SQL/DS, servindo como sistema e repositório de informações de outras empresas.

Estes sistemas somente nasceram a partir da insistência de um jornal técnico em utilizar BNF para SQL e este jornal publicou tal artigo. BNF é o conjunto de sintaxes de linguagem de computador que explica exatamente como cada comando interage com os outros comandos e o que pode ou não ser realizado, como os comandos são formados em assim por diante. Por causa da publicação deste artigo, empresas puderam utilizá-lo para modelar seus próprios sistemas, os quais seriam 100% compatíveis com o sistema da IBM.

2.3.3.3 Evolução

O *software* de banco de dados relacionais foi sendo refinado durante a década de 80. Isso deveu-se ao *feedback* (retorno) que os usuários destes sistemas faziam, devido ao desenvolvimento de sistemas para novas indústrias e ao aumento do uso de computadores pessoais e sistemas distribuídos.

Desde sua chegada, os bancos de dados têm tido aumento nos dados de armazenamento, desde os 8 MB (*Megabytes*) até centenas de *Terabytes* de dados em listas de e-mail, informações sobre consumidores, sobre produtos, vídeos, informações geográficas etc. Com este aumento de volume de dados, os sistemas de bancos de dados em operação também sofreram aumento em seu tamanho.

2.3.3.4 Orientação a objetos

Em meados da década de 80 tornou-se óbvio que existiam várias áreas onde bancos de dados relacionais não eram aplicáveis, por causa dos tipos de dados envolvidos. Estas áreas incluíam medicina, multimídia e física de energia elevada, todas com necessidades de flexibilidade em como os dados seriam representados e acessados.

Este fato levou ao início de pesquisas em bancos de dados orientados a objetos, os quais os usuários poderiam definir seus próprios métodos de acesso aos dados e como estes seriam representados e acessados. Ao mesmo tempo, linguagens de programação orientadas a objetos (*Object Oriented Programming* - POO) tais como C++ começaram a surgir na indústria.

No início de 1990, temos a aparição do primeiro Sistema de Gerenciamento de Banco de Dados Orientado a Objetos, através da companhia *Objectivity*. Isso permitiu com que usuários criassem sistemas de banco de dados para armazenar resultados de pesquisas como o CERN (maior laboratório que trabalha com partículas físicas em pesquisas nucleares - europeu) e SLAC (Centro de Aceleração Nuclear - norte-americano), para mapeamento de rede de provedores de telecomunicações e para armazenar registros médicos de pacientes em hospitais, consultórios e laboratórios.

2.3.3.5 Modelagem de dados

O conhecimento acerca dos modelos entidade-relacionamento (MERs) possibilita a criação de pequenos e grandes projetos lógicos de bancos de dados e de modelagens de alta qualidade, propiciando o sucesso dos projetos de software ou aplicação.

2.3.3.6 Modelagem e projeto de banco de dados

Um dos momentos mais críticos no processo de desenvolvimento de um software é a modelagem de banco de dados, pois o produto deve atingir os objetivos estabelecidos pelo requisitante. Segundo Heuser (2009), previamente à construção de bancos de dados, são utilizados padrões em textos e gráficos para modelar, sendo propostos três níveis de abstração de dados: modelo conceitual, modelo lógico e modelo físico. Como muitos usuários de banco de dados são leigos nas técnicas de informática, faz-se necessário simplificar em um projeto a

sua estrutura, para oferecer uma visão geral dos dados com os aspectos de interesse, possibilitando bancos de dados flexíveis (COUGO, 1997).

Os requisitos podem ser descritos graficamente por diagramas, com declaração sobre as funções que o sistema deve oferecer de forma abstrata de alto nível. Leva-se também em consideração a engenharia de requisitos, que é um processo que engloba todas as atividades que contribuem para a elaboração de um documento, com todos os requisitos, e para a sua manutenção. A etapa de engenharia de requisitos é a fase de descobrir, analisar e verificar funções e restrições (GIMENES; HUZITA, 2005).

Um erro durante a modelagem compromete a usabilidade do sistema final e acarreta a necessidade de retrabalho, o que aumenta o custo do processo de desenvolvimento. Para que isso não ocorra, a seguir serão apresentados os passos fundamentais do processo de modelagem de um banco de dados, conforme as fases apresentadas na figura 7.

Figura 7 – Etapas de modelagem de dados

Fonte: Adaptado de Cougo(1997)

2.3.3.6.1 Identificação do problema (levantamento de requisitos)

Nesta etapa, é realizado um estudo detalhado das atividades em questão. Quando não há conhecimento prévio sobre o negócio, entrevistas podem levantar informações relevantes sobre as necessidades dos futuros usuários. Os administradores de dados se reúnem com os usuários para entender e documentar seus requisitos.

Os requisitos são a base de todos os produtos de software. Sua elucidação, seu gerenciamento e seu entendimento são problemas comuns a todas as metodologias de desenvolvimento. Segundo Pressman e Maxim (2011), a tarefa de análise de requisitos é um processo de descoberta, refinamento, modelagem e especificação. A análise de requisitos proporciona ao projetista de software uma representação da informação e da função, que pode ser traduzida no projeto procedimental, arquitetônico e de dados, oferecendo ao desenvolvedor e ao cliente os critérios para avaliar a qualidade logo que o sistema for construído.

2.3.3.6.2 Modelagem conceitual (alto nível)

A modelagem conceitual é a representação que considera exclusivamente o ponto de vista do usuário criador dos dados, levando em consideração fatores técnicos para sua implementação. O nível conceitual especifica como os dados são armazenados e relacionados, independentemente de como serão implementados no banco de dados. Para Heuser (2009, p. 25), "[...] a técnica de modelagem conceitual mais difundida é a abordagem entidade-relacionamento. Nessa técnica, um modelo conceitual é usualmente representado através de um diagrama". O MER utiliza elementos gráficos para descrever o modelo de dados de uma aplicação com alto nível de abstração (CALIARI, 2007), identificando entidades, atributos e relacionamentos. Peter Chen, em 1976, idealizou uma notação para realizar a modelagem de dados para ambientes relacionais.

2.3.3.6.3 Modelagem lógica (representativa ou de implementação)

O modelo lógico só deve ser inicializado após a conclusão do modelo conceitual. Diferentemente do modelo conceitual, o modelo lógico será criado com base em um tipo de banco de dados, como SQL *Server, Oracle,* MySQL, dentre outros.

Muitos analistas não aceitam que a etapa do modelo conceitual seja importante, acreditando que ela é desnecessária. Devido aos prazos curtos dos projetos, tais analistas não criam o modelo conceitual e iniciam o projeto com o desenvolvimento do modelo lógico. Porém, no fim, muitos se dão conta de que nem todo requisito ou solicitação ficou completo ou foi atendido corretamente, o que poderia ser facilmente criado e interpretado na elaboração do modelo conceitual.

De acordo com Heuser (2009) e Machado (2014), o modelo lógico descreve e mapeia as estruturas que estarão presentes no banco de dados, de acordo com as características da abordagem. Evitam-se:

- muitas tabelas;
- tempo longo de resposta nas consultas e atualizações de dados;
- desperdício de espaço;
- muitos controles de integridade no banco de dados;
- muitas dependências entre dados.

2.3.3.6.4 Modelagem física (baixo nível)

O modelo físico é concebido por meio do modelo lógico. É nesse modelo que serão definidos os tipos de dados que serão armazenados, e ocorre a implementação da estrutura lógica em um sistema gerenciador de banco de dados (SGBD), que administra fisicamente os dados armazenados. Esse modelo se resume à SQL, que é a linguagem necessária para gerenciar um banco de dados relacional (OLIVEIRA, 2002). Nele, são detalhados os componentes da estrutura física do banco, como tabelas, campos, tipos de valores, índices etc. Entram em cena os detalhes técnicos do projeto, atendendo à necessidade do cliente e já implantando a política de cópia e segurança. Nessa fase, há a geração das instruções em código SQL, que vão criar a base de dados do sistema. Por isso, nesse ponto, a tecnologia aplicada assume lugar primordial, pois a parte de negócios já foi definida e estabelecida.

2.3.3.7 Introdução aos SGBD

Um sistema de gerenciamento de banco de dados SGBD consiste em uma coleção de dados inter-relacionados e em um conjunto de programas para acessá-los. Um conjunto de dados, normalmente referenciado como banco de dados, contém informações sobre uma empresa particular, por exemplo. O principal objetivo de um SGBD é prover um ambiente que seja adequado e eficiente para recuperar e armazenar informações de banco de dados.

Os sistemas de banco de dados são projetados para gerenciar grandes grupos de informações. O gerenciamento de dados envolve a definição de estruturas para armazenamento de informação e o fornecimento de mecanismos para manipulá-las. Além disso, o sistema de banco de dados precisa fornecer segurança das informações armazenadas, caso o sistema dê problema, ou contra tentativas de acesso não-autorizado. Se os dados devem ser divididos entre diversos usuários, o sistema precisa evitar possíveis resultados anômalos.

A importância das informações na maioria das organizações e o consequente valor dos bancos de dados têm orientado o desenvolvimento de uma grade corpo de conceitos e técnicas para o gerenciamento eficiente dos dados.

2.3.3.7.1 MySQL

O programa MySQL é um sistema de gerenciamento de banco de dados relacional que utiliza a linguagem de consulta estruturada SQL como interface de acesso e extração de informações do banco de dados em uso. O MySQL é um dos sistemas de gerenciamento de bancos de dados mais popular e usado no mundo. É rápido, multitarefa e multiusuário.

O MySQL originou-se na Suécia e as ideias para seu desenvolvimento surgiram em 1979 de dois suecos (David Axmark e Allan Larsson e um finlandês (Michael Widenius, também conhecido como Michael "Monty" Widenius), que trabalhavam juntos como programadores numa pequena empresa chamada TcX, que desenvolveu uma ferramenta de banco de dados (não SQL) para o gerenciamento de grandes tabelas denominada Unireg, utilizada para geração de relatórios.

O programa Unireg foi escrito em linguagem BASIC para ser executada em computadores de 16 Kbytes de memória RAM (*Randon Access Memory*) que operavam a 4 MHz. Posteriormente essa mesma ferramenta foi reescrita em linguagem C para computadores com sistema operacional UNIX.

Em 1994, Monty Widenius iniciou o desenvolvimento de um gerenciador de banco de dados de código aberto baseado nos programas Unireg e mSQL, que não era muito bom, com tabelas grandes, mas serviu de aprendizado para o surgimento de um novo produto.

A partir dessa influência os três amigos iniciaram, em 1995, o desenvolvimento do MySQL pela empresa MySQL AB, e o lançamento da primeira versão oficial ocorreu no ano de 1996, codificada com o número 3.11.1. Em janeiro de 2008 o programa MySQL foi comprado pela empresa *Sun Microsystems*, Inc. pelo valor de 1 bilhão de dólares.

O MySQL a cada dia torna-se um produto apreciado por várias empresas, entidades e pessoas, pois possui um servidor confiável, rápido e de fácil utilização, que pode ser utilizado com grandes ban cos de dados, considerando inclusive aplicações voltadas para a Internet.

Aliás, parte de seu sucesso é devido à fácil integração com a linguagem de script PHP. São muitas as empresas que usam o produto MySQL em aplicações críticas, como ferramenta de trabalho para a manutenção de bancos de dados, o MySQL possui as seguintes características: portabilidade, rodando em diversas plataformas computacionais, como AIX 4.x e 5.x com subprocessador nativo, Amiga, BSDI 2.x com o arquivo "MIT-pthreads", BSDI 3.0, 3.1 e 4.x com subprocessador nativo, Digital Unix 4.x com subprocessador nativo, FreeBSD

2.x com o arquivo “MIT-pthreads”, FreeBSD 3.x e 4.x com subprocessador nativo, FreeBSD 4.x com “LinuxThreads” instalado, HP-UX 10.20 com o arquivo “DCE threads” e “MIT-pthreads” ins- talados, HP-UX 11.x com subprocessador nativo, Linux 2.0+ com “LinuxThreads 0.7.1+” e “glibc 2.0.7+” instalados para várias arquiteturas de CPU, Mac OS X, NetBSD 1.3/1.4 Intel e NetBSD 1.3 Alpha (requer GNU make), Novell NetWare 6.0, OpenBSD que é a versão 2.5 com subprocessador nativo, OpenBSD menor que a versão 2.5 com o arquivo “MIT-pthreads” ins- talado, OS/2 Warp 3 com FixPack 29, OS/2 Warp 4 com FixPack 4, SCO OpenServer 5.0.X com uma versão do arquivo “FSU Pthreads” mais recente possível, SCO UnixWare 7.1.x, SCO Openserver 6.0.x, SGI Irix 6.x com subprocessador nativo, Solaris 2.5 e posteriores com sub- processador nativo para a plataforma SPARC e plataforma x86, SunOS 4.x com o arquivo “MIT-pthreads”, Tru64 Unix e MS-Windows (9x, Me, NT, 2000, XP, e 2003); compatibilidade com os wwwdrivers ODBC, JDBC e .NET, além de possuir módulos de interfaceamento com as linguagens de programação: Java, C, C++, Python, Perl, PHP e Ruby; facilidade de uso, excelente desempenho e estabilidade, exigindo poucos recursos de hardware, além de possuir uma versão baseada na filosofia software livre.

Como curiosidade, o logotipo do gerenciador de banco de dados MySQL é a imagem de um golfinho de nome Sakila que, segundo os desenvolvedores do programa, representa a imagem dos valores da empresa MySQL AB, e também a imagem do sistema de gerenciamento de banco de dados em si, uma vez que o programa MySQL é uma ferramenta rápida, precisa, com grande potencial e facilidade de uso.

O nome do produto MySQL sugere que a linguagem de consulta usada pelo servidor de banco de dados é a SQL. O programa MySQL segue boa parte dos padrões SQL normalizados e propostos segundo a entidade ANSI, mas também acrescenta outros recursos particulares que não são previstos segundo o padrão ANSI SQL.

2.3.4. Desenvolvimento de software

2.3.4.1 Conceitos de Desenvolvimento de Software

Um grande número de nossas atividades diárias sãoé realizadas com o apoio de software. Exemplos dessas atividades são:

- Busca de páginas na internet mais relacionadas com um conjunto de palavras digitadas em uma máquina de busca.
- Cálculo do custo de uma conta telefônica para as chamadas realizadas no mês.

- Cálculo do salário a ser pago a um funcionário, conforme sua função, número de horas trabalhadas e valores a serem descontados.
- Consulta de saldo bancário em caixas automáticos.
- Matrícula em disciplinas para o próximo semestre letivo.
- Monitoramento de frequência de batimentos cardíacos e distância percorrida durante exercícios físicos.

Realizar essas tarefas utilizando Computação nada mais é do que encontrar uma solução que possa ser executada pelo dispositivo computacional (*hardware*) utilizado.

O que um software faz é receber alguns dados ou valores como entrada, realizar um conjunto de operações, que podem incluir cálculos, e retornar como resultado, novos dados ou uma sequência de ações.

2.3.4.1.1 *Software* e Programa

Um programa ou *software* nada mais é do que uma sequência de passos ou instruções descritos por um algoritmo, que, quando executados, fazem com que o computador realize uma tarefa. Um algoritmo descreve a sequência de passos de um programa em uma linguagem próxima de um idioma, como o português. Normalmente, o programa está escrito em uma linguagem de programação e faz uso de diferentes estruturas para manipular os dados e processá-los. Estrutura de dados é uma forma de representar os dados para possibilitar sua manipulação adequada por programas.

Muitas vezes, alguém que não escreveu o software precisa corrigi-lo ou atualizá-lo. Para isso, é importante que o *software* seja bem descrito por quem o fez. Essa descrição, ou documentação, descreve os componentes do *software*, suas estruturas de dados, o funcionamento de seus programas e como o *software* pode ser utilizado.

Um *software* tanto pode ser comercializado como usado de forma gratuita. Uma organização sem fins lucrativos, a Fundação para o *Software* Livre, ou *Free Software Foundation*, defende que não deve haver restrições para criar, distribuir e modificar software.

Outro conceito importante, muitas vezes confundido com *software* livre, é o de código aberto, defendido pela Iniciativa para o Código Aberto, ou *Open Software Initiative*, que prega que qualquer pessoa deve ter acesso ao código-fonte de um programa. Nesse caso, o desenvolvedor do código pode definir as condições para uso do *software*, ou seja, o *software* pode ser ou não livre.

Movimentos em defesa do software livre e do código aberto têm atraído o apoio de muitos programadores e de pesquisadores em todo o mundo.

Para que o *hardware* do computador funcione é necessária a existência do *software* – instruções eletrônicas armazenadas, conhecidas como programas. Costumamos dizer que o *software* dá vida à máquina. Por meio de instruções contidas em *software* (programas) o computador é capaz, por exemplo, de acionar um disco, determinar a gravação de um dado, enviar um comando para a impressora, imprimir um texto, entre outras coisas.

Esses programas compõem o que se chama de parte logica do computador, a qual é intangível, não podendo ser tocada, como é o caso do *hardware*. Quando estão sendo usados – rodando ou executados –, vários comandos são enviados para acionar componentes físicos da máquina.

Existem vários tipos de programas disponíveis, os quais podem se diferenciar por suas características, finalidade e nível em que atuam. Podem ser divididos em duas categorias principais:

- Básico: o mais importante dos softwares básicos é o chamado sistema operacional, que é constituído por um conjunto de outros programas integrados contendo instruções que trabalham para coordenar todas as atividades entre o computador e os recursos de *hardware*.
- Aplicativo: programas mais interativos com o usuário que permitem executar tarefas específicas de um determinado problema.

O gerenciamento de dispositivos inclui comandos para ligar e desligar o computador, acessar discos, iniciar e carregar programas para a memória, gerenciar e interpretar os comandos de entrada (teclado e mouse, por exemplo), conectar-se à Internet, atualizar-se, entre outros. Geralmente o sistema operacional é instalado no disco rígido do computador pessoal.

Considerando que o sistema operacional se destina basicamente a gerenciar o computador, outros programas são necessários para atender as necessidades específicas do usuário, que incluem, por exemplo, calcular, enviar mensagem, escrever, navegar, etc. Essas necessidades deram origem ao que chamamos de software aplicativo, cuja finalidade é ajudar os usuários a desempenhar uma tarefa específica.

É possível encontrar aplicativos para praticamente todas as áreas do conhecimento. Considerando que a diversidade é muito grande, uma classificação geral quanto à natureza do software aplicativo é importante para facilitar a diferenciação e o entendimento. Podemos classificar os aplicativos em:

- Aplicações comerciais;
- Utilitários;
- Aplicações pessoais;
- Aplicações de entretenimento.

2.3.4.2 Tipos de aplicações

2.3.4.2.1 Aplicações comerciais

Aplicações comerciais são programas ou um conjunto de aplicativos desenvolvi- dos inicialmente para o ambiente comercial, mas que se popularizaram tanto que passaram a ser amplamente usados por usuários comuns. Neste contexto, podemos citar os editores de texto, os editores de apresentação, os editores de imagem e as planilhas eletrônicas. Exemplos desses aplicativos são o *Microsoft Office* (*Word, Excel e Powerpoint*), o *BrOffice* (*Writer, Impress* e o *Calc*), o *Photoshop* e o *Gimp*.

2.3.4.2.2 Utilitários

Utilitários são os aplicativos destinados a gerenciar e realizar a manutenção do computador do cliente. Entre suas funções, podemos incluir a melhora no desempenho do computador (Defrag do *Windows*), a redução no consumo de energia, a realização da cópia de segurança (*backup*), a personalização da área de trabalho, as ferramentas administrativas, o verificador de integridade física do disco, o gerenciador de arquivos, etc.

2.3.4.2.3 Aplicações pessoais

Aplicações pessoais são os aplicativos que tem como objetivo auxiliar nas tarefas pessoais do usuário final. Um organizador simples de fotos, uma agenda eletrônica que permite ao usuário classificar os seus contatos, um aplicativo de chat ou de mensagens instantâneas que permite troca de mensagens entre usuários são exemplos dessa categoria.

2.3.4.2.4 Aplicações de entretenimento

Aplicações de entretenimento são os aplicativos utilizados para proporcionar lazer e diversão, normalmente incluindo jogos que também podem ser usados com finalidades educacionais. Caracterizam-se por dispor de muitos recursos interativos, usando estratégias e simulações, visando ao desenvolvimento do intelecto, do raciocínio logico e até mesmo da socialização em caso de jogos coletivos em que múltiplos usuários colaboram para atingir um determinado fim. Outras classificações de software podem ser encontradas em literaturas específicas, envolvendo especificidades e usos mais restritos. Entre essas, podemos citar as linguagens de programação usadas para o desenvolvimento de qualquer software, aplicativos para simulação computacional, aplicativos para diagnósticos, entre outros.

Software, em geral, tem outra classificação importante quanto ao seu uso: *software* proprietário e *software* livre. O *software* proprietário é aquele que tem sua distribuição ou alteração limitada por questões de registro ou patente, o que implica na aquisição de uma licença para uso. Exemplos desse tipo de *software* são:

- Sistema operacional *Microsoft Windows*
- *Microsoft Office*
- Mac OS
- Adobe *Photoshop*

Já o *software* livre segue uma filosofia oposta, permitindo que qualquer programa possa ser utilizado, copiado, alterado e redistribuído sem restrições, conforme a definição da *Free Software Foundation*. Exemplos de *software* livre são o sistema operacional Linux e o *LibreOffice*. Quanto à liberdade de uso do *software*, é importante conhecer os seus diversos tipos de distribuição. Entre os principais estão o *freeware*, o *shareware*, o *trial* e o *Demo*. O *freeware*, mesmo sendo um software proprietário, é disponibilizado para uso gratuito e não pode ser modificado. O *shareware* é um software disponibilizado gratuitamente, mas apenas por um período ou com algumas funções não disponíveis, o que implica pagamento pela sua licença após o fim desse período.

Um *software* lançado com uma versão de teste em que apenas algumas funções são disponibilizadas é chamado de *Trial*. O objetivo é oferecer ao usuário uma oportunidade para experimentar o software – normalmente o usuário pode utilizar o *software* em um período de 30 dias para saber se ele atende as suas necessidades. Da mesma forma, o software considerado

Demo oferece uma versão de demonstração bastante parecida com a do *Trial*, permitindo o uso do software por um tempo determinado ou com apenas algumas funções disponíveis.

Essa programação do *software* é realizada pelo programador de computadores, nome dado ao profissional de informática que domina, além da lógica de programação e algoritmos, uma linguagem de programação, um tipo de *software* projetado para ser compreensível e programável por humanos e compilado e traduzido para a linguagem que o computador compreende, ou seja, a linguagem de máquina.

Existem várias linguagens de programação. Algumas são mais adequadas para um tipo de programa, como os executáveis no *desktop*, e outras para sistemas como a Internet. Dentre elas, as mais conhecidas são as linguagens:

- C;
- Java;
- PHP;
- C++;
- JavaScript;
- Python;
- Pascal;
- Delphi;
- Visual Basic;
- C#;
- Assembly.

Enquanto Java, PHP e Python são linguagens destinadas ao desenvolvimento de aplicativos para rodarem na *Web*, outras como C, C++ e Delphi são próprias para o desenvolvimento de aplicativos para o PC. Esses aplicativos raramente são desenvolvidos em uma linguagem como Assembly, por exemplo, que é muito próxima da linguagem de máquina e´ utilizada frequentemente em *software* embarcado ou *software* embutido para aumentar a velocidade de execução ou diminuir o espaço necessário de armazenamento.

A linguagem de máquina, traduzida pela linguagem de programação, contém instruções para o *hardware* na forma de números, que é tipicamente a unidade e a linguagem que o computador conhece no seu nível mais baixo. É, portanto, fundamental ao programador conhecer como o computador reconhece e trabalha com dados.

2.3.4.3 O Processo de desenvolvimento de software

Um processo de desenvolvimento de *software* pode ser visto como um conjunto de atividades organizadas, usadas para definir, desenvolver, testar e manter um *software.*

Existem diversos processos de desenvolvimento de *software*, no entanto há algumas atividades básicas comuns à grande parte dos processos existentes, como: levantamento de requisitos; análise de requisitos; projeto; implementação; testes; implantação.

2.3.4.3.1 Levantamento de Requisitos

Esta atividade tem como objetivo, compreender o problema, dando aos desenvolvedores e usuários, a mesma visão do que deve ser construído para resolução do problema. Desenvolvedores e clientes, em conjunto, buscam levantar e priorizar as necessidades dos futuros usuários do *software* (necessidades essas denominadas como requisitos).

O levantamento de requisitos é a etapa mais importante, no que diz respeito ao retorno de investimentos no projeto. Vários projetos são abandonados pelo baixo levantamento de requisitos, ou seja, membros da equipe não disponibilizaram tempo suficiente para essa fase do projeto, em compreender as necessidades dos clientes em relação ao sistema a ser desenvolvido.

E como um sistema de informações geralmente é utilizado para automatizar processos de negócio em uma organização, esses processos da organização devem ser bem compreendidos para que o restante das atividades do processo de desenvolvimento flua de acordo com as reais necessidades do cliente.

2.3.4.3.2 Análise de Requisitos

Esta etapa, também chamada de especificação de requisitos, é onde os desenvolvedores fazem um estudo detalhado dos dados levantados na atividade anterior. De onde são construídos modelos a fim de representar o sistema de *software* a ser desenvolvido.

O interesse nessa atividade é criar uma estratégia de solução, sem se preocupar como essa estratégia será realizada, ou seja, utilizar as necessidades dos clientes, depois de compreendido o problema, para resolução do problema solicitado. Assim é necessário definir o que o sistema deve fazer, antes de definir como o sistema irá fazer.

O que acontece com frequência, é quando as equipes de desenvolvimento partem para a solução do problema do *software*, sem antes ter definido completamente o problema em questão. Nesta fase deve-se então realizar a validação e verificação dos modelos construídos, antes de partir para solução do problema.

- Validação: tem por objetivo, assegurar que o sistema de software está atendendo às reais necessidades do cliente;
- Verificação: verifica se os modelos construídos na análise estão em conformidade com os requisitos do cliente.

2.3.4.3.3 Projeto

Nesta fase é que deve ser considerado, como o sistema funcionará internamente, para que os requisitos do cliente possam ser atendidos. Alguns aspectos devem ser considerados nessa fase de projeto do sistema, como: arquitetura do sistema, linguagem de programação utilizada, Sistema Gerenciador de Banco de Dados (SGBD) utilizado, padrão de interface gráfica, entre outros.

No projeto é gerada uma descrição computacional, mencionando o que o *software* deve fazer, e deve ser coerente com a descrição realizada na fase de análise de requisitos.

O projeto possui duas atividades básicas: projeto da arquitetura (ou projeto de alto nível), e projeto detalhado (ou projeto de baixo nível).

Em um processo de desenvolvimento orientado a objetos, o projeto da arquitetura normalmente é realizado por um arquiteto de software. O projeto da arquitetura visa distribuir as classes de objetos relacionados do sistema em subsistemas e seus componentes, distribuindo também esses componentes pelos recursos de hardware disponíveis.

Já no projeto detalhado, são modeladas as relações de cada módulo com o objetivo de realizar as funcionalidades do módulo. Além de desenvolver o projeto de interface com o usuário e o projeto de banco de dados.

2.3.4.3.4 Implementação

Nessa etapa, o sistema é codificado a partir da descrição computacional da fase de projeto em uma outra linguagem, onde se torna possível a compilação e geração do código-executável para o desenvolvimento software.

Em um processo de desenvolvimento orientado a objetos, a implementação se dá, definindo as classes de objetos do sistema em questão, fazendo uso de linguagens de programação como, por exemplo: Delphi (Object Pascal), C++, Java, etc. Pode-se também utilizar na implementação ferramentas de software e bibliotecas de classes preexistentes para

agilizar a atividade, como também o uso de ferramentas CASE, que dinamizam o processo de desenvolvimento, nas várias atividades, onde inclui-se geração de código-fonte, documentação, etc.

2.3.4.3.5 Testes

Diversas atividades de testes são executadas a fim de se validar o produto de *software*, testando cada funcionalidade de cada módulo, buscando, levando em consideração a especificação feita na fase de projeto. Onde o principal resultado é o relatório de testes, que contém as informações relevantes sobre erros encontrados no sistema, e seu comportamento em vários aspectos. Ao final dessa atividade, os diversos módulos do sistema são integrados, resultando no produto de software.

2.3.4.3.6 Implantação

Por fim a implantação compreende a instalação do software no ambiente do usuário. O que inclui os manuais do sistema, importação dos dados para o novo sistema e treinamento dos usuários para o uso correto e adequado do sistema. Em alguns casos quando da existência de um *software* anterior, também é realizada a migração de dados anteriores desse *software*.

2.3.4.4 Métodos ágeis de desenvolvimento de software

As mudanças na economia, nos serviços concorrentes e nos novos produtos tem sido cada vez mais rápida e os negócios precisam responder com a mesma velocidade, para não perder essas novas oportunidades. Como o *software* está presente em todas as operações do negócio, é essencial que novos *softwares* sejam desenvolvidos rapidamente para não perder essas novas oportunidades e também responder às pressões da competitividade. Pensando em formas de desenvolver softwares com maior velocidade e mantendo a qualidade e a satisfação dos clientes, em 2001, Kent Beck e outros 16 desenvolvedores assinaram o “Manifesto para o

desenvolvimento Ágil de *Software*", que deu origem aos processos de desenvolvimento para criar software útil rapidamente. Geralmente, são processos iterativos nos quais a especificação, o projeto, o desenvolvimento e o teste são intercalados. Usando esse método, o software não é desenvolvido e disponibilizado integralmente, e sim uma série de incrementos e a cada novo incremento uma nova funcionalidade do sistema. Existem muitas abordagens para o desenvolvimento de software rápido, e elas compartilham de características fundamentais:

- os processos de especificação, projeto e implementação são concorrentes;
- o sistema é desenvolvido em uma série de incrementos;
- as interfaces com o usuário são desenvolvidas usando um sistema de desenvolvimento interativo.

Para garantir a agilidade no desenvolvimento do software, veja a seguir os princípios de agilidade. Embora nem todo modelo de processo ágil aplique todos esses métodos, tais princípios definem um espírito ágil:

- a maior prioridade é satisfazer o cliente;
- acolha bem os pedidos de alterações, mesmo se o projeto estiver atrasado;
- faça entregas frequentes do software em funcionamento;
- a equipe comercial e os desenvolvedores devem trabalhar em conjunto;
- mantenha a equipe motivada;
- mantenha conversas abertas e presenciais com a equipe;
- software em funcionamento é a principal medida do progresso;
- os processos ágeis promovem desenvolvimento sustentável;
- mantenha a excelência técnica;
- simplicidade é essencial;
- as melhores arquiteturas, requisitos e projetos emergem de equipes que se auto-organizam;
- a equipe tem sempre foco na eficiência.

Das metodologias ágeis de desenvolvimento de *software*, podemos citar, entre as mais usadas, XP, Scrum e OpenU.

2.3.4.5 Qualidade de *software*

A palavra qualidade parece ter um significado bastante óbvio, contudo, trata--se de um elemento complexo e de difícil mensuração, dado que a qualidade é relativa e pode assumir diversos valores, de acordo com a pessoa que a observa. Nesse sentido, a qualidade do *software*

pode ser medida de acordo com o quanto ele está em conformidade com o que o cliente solicitou. Ou seja, mensura-se se o *software* está em conformidade com os requisitos funciona e não funcionais especificados explicitamente pelo cliente, conforme leciona Basu (2015).

Embora a qualidade do *software* seja uma prática recente, que ganhou evidência com o advento da tecnologia, a qualidade, no geral, é uma preocupação bem antiga. Existem registros de que há mais de quatro mil anos os egípcios estabeleceram um padrão de medida para realizar seus trabalhos de forma apurada, chamado de cúbito, que equivalia ao tamanho do braço do faraó reinante, conforme Koscianski e Soares (2007). Entretanto, mesmo a qualidade sendo um conceito tão antigo, os projetos de software contam com vários desafios para entregar o *software* em perfeito funcionamento, devido a uma série de fatores — em especial, a complexidade. Isso porque construir um sistema envolve diversas habilidades, como comunicação e interpretação, para conseguir entender o que o cliente deseja, além de habilidades específicas para as etapas de programação, análise da qualidade, entre outras, que são de grande complexidade.

Dada a multidisciplinariedade envolvida no processo de desenvolvimento de *software* e a complexidade de todo o processo, o segmento de qualidade se divide em duas áreas relacionadas, porém distintas: qualidade de processos e qualidade de produtos. A área de qualidade de processos trata da organização sistemática dos processos da empresa, visando ao melhor andamento dos projetos de desenvolvimento de sistemas, otimizando o tempo, tornando os processos repetitivos e evitando problemas em situações críticas para os projetos, por exemplo: estimativa, custo, entrada e saída de recursos humanos.

Ao buscar garantir a qualidade de um software, estamos diante do desafio de estabelecer uma cultura de não tolerância a erros, por meio de processos que objetivem inibir ou impedir falhas, segundo Bartié (2002). É a área de qualidade de processos que se responsabiliza pela definição da metodologia ou do ciclo de vida de *software* que a equipe utilizará, podendo optar, por exemplo, por trabalhar com métodos ágeis ou métodos tradicionais. Independentemente da metodologia escolhida, o importante é que um processo seja seguido, para que a equipe tenha um padrão para se basear. Isso melhora a comunicação e influencia na qualidade dos produtos desenvolvidos pela equipe.

A área de qualidade de produtos tem por objetivo garantir a qualidade do produto tecnológico gerado durante o ciclo de desenvolvimento. Para esse fim, são realizadas atividades com o objetivo de estressar as funcionalidades do sistema, identificando o comportamento dele nesse contexto. Essas atividades são chamadas de testes de software. Essa área possui algumas

divisões, sendo a mais importante a subdivisão entre testes que fazem uso do código-fonte do programa, chamados de caixa branca, e testes que não fazem uso do código--fonte do programa, chamados de caixa preta. Além disso, os testes podem ser feitos de forma manual ou automatizada e fazendo uso das mais diversas técnicas, conforme Bartié (2002).

A eficiência de um processo de testes é afetada diretamente por alguns fatores, que devem ser considerados para evitar problemas nas organizações, aponta Bartié (2002): falta de planejamento das atividades de testes, ausência de testes que validem funcionalidades antigas e ausência de processos de automação de testes.

Uma terminologia bastante importante e comum na área de testes de qualidade de *software* é o conceito de *bug*, que abrange erros, defeitos e falhas. Por curiosidade, o termo *bug* corresponde à palavra inseto em inglês e começou a ser utilizado justamente quando um inseto causou uma falha em um equipamento. É importante entender que os termos erro, defeito e falha se referem a coisas distintas. Defeito é um comportamento inesperado de um produto. O defeito está em uma parte do produto e, em geral, refere-se a uma funcionalidade que está implementado no código de maneira incorreta. Erro é aquilo que foi cometido pelo programador e que gerou um código defeituoso, enquanto a falha se dá quando o programa defeituoso é executado e interfere no funcionamento do sistema para o usuário final. Falhas também podem ocorrer por fatores externos ao programa, como corrupção de bases de dados ou invasões de memória por outros programas, conforme Koscianski e Soares (2007).

A linha de código que calcula o valor para a variável c pode apresentar um problema, dado que não é feita uma verificação para validar se o valor de a é 0. Dessa forma, pode acontecer um erro ao tentar realizar uma divisão por zero. O comportamento anormal do programa, que provavelmente gera um bug ou uma interrupção da execução, é provocado pela divisão b/a. Em um primeiro momento, podemos dizer que essa linha de código é defeituosa.

Existe, entretanto, outra hipótese: o defeito pode estar na rotina *input* (). Imagine que a especificação dessa rotina estabeleça que ela não deve jamais retornar um valor nulo. Nesse caso, o erro foi cometido pelo programador responsável por essa rotina. Essa segunda hipótese é bastante razoável, pois, para a maioria dos programas, não é recomendado preceder cada operação de divisão com um teste *if*. Nesse caso, um erro cometido pelo programador na rotina *input* fez com que o programa apresentasse um defeito ao executar a divisão de a por b.

Inúmeros são os benefícios que as empresas podem ter ao demandar uma atenção especial para a área de qualidade de *software*. Os benefícios vão muito além de valores financeiros, podendo estar relacionados inclusive com evitar transtornos legais ou preservar vidas.

É por isso que o controle de qualidade é totalmente importante e benéfico nos dias atuais. A qualidade de *software* possui diversos benefícios que ajudam os usuários a não sofrerem com falhas de software e as empresas a oferecerem produtos melhores, impedindo até mesmo grandes catástrofes. Vamos verificar alguns dos benefícios da qualidade de *software*, com base em Basu (2015):

- economiza dinheiro;
- impede emergências corporativas catastróficas;
- inspira a confiança do cliente;
- mantém o nível de experiência do usuário elevado;
- traz mais lucro;
- aumenta a satisfação do cliente;
- promove organização, produtividade e eficiência.

Existe uma regra que vigora desde os princípios do teste de software, chamada regra 10 de Myers. Essa regra estipula que o custo de encontrar um defeito no sistema aumenta 10 vezes a cada etapa do processo em que esse erro avançar, conforme aponta Myers (1979). Essa regra é mais aplicada para o contexto de projetos com ciclo de vida em cascata, em que as etapas são executadas sequencialmente, dado que, em equipes ágeis, não existe essa divisão exata de etapas.

2.3.5 Versionamento

Um sistema de controle de versão (VCS – *Version Control System*) é uma solução utilizada para gerenciar a mudança em documentos programas. A proposta é fazer com que as alterações realizadas, por um membro de uma eventual equipe de desenvolvimento, estejam presentes para outros componentes, sem a necessidade de análises complexas para prover integrações, bem como evitar conflitos.

Segundo Moura (2013, p.19):

> Os repositórios, também conhecidos como Sistemas de Controle de Versão (SCV) ou simplesmente versionadores (e.g., CVS, *Subversion*, Git), são responsáveis por registrar e facilitar o controle da evolução do *software*. Tal evolução consiste em mudanças que podem ser de três tipos: alterações, inserções e remoções de linhas ou arquivos do código fonte, seja para acrescentar novas características ao programa ou para corrigir problemas conhecidos (*bugs*). Além disso, há recursos mais avançados como ramificar um projeto de software ou unificar ramos de desenvolvimento.

O quadro 12, apresentado abaixo, demonstra os problemas que podem ocorrer com a falta de um sistema de controle de versão.

Quadro 12 – Problemas causados pela falta de um sistema de controle de versões

Problemas	**O que ocorre sem um sistema de controle de versões**
Histórico	não há o registro da evolução do projeto e das alterações sobre cada arquivo; sem essas informações não se sabe quem fez o que, quando e onde; também é impossível revisitar versões sempre que desejado, pois os arquivos foram ~~sobreescritos~~sobrescritos e não há possibilidade de reversão.
Colaboração	não há a possibilidade que vários desenvolvedores trabalhem em paralelo sobre os mesmos arquivos, resultando na probabilidade de que um sobrescreva o código de outro, o que pode acarretar no surgimento de defeitos e perda de funcionalidades.
Variações no Projeto	não há a possibilidade de manter linhas diferentes de evolução do mesmo projeto; por exemplo, não é mantida uma versão 1.0 enquanto a equipe prepara uma versão 2.0.

Fonte: Adaptado de Palestino *apud* Dias (2015, p.30)

A utilização de um sistema de controle de versão, permite aos desenvolvedores, o acompanhamento de alterações desde as versões mais antigas, assim como a possibilidade de detectar e mesclar atualizações em mesmos arquivos, além de identificar conflitos. A solução dispõe de um repositório, que permite o acesso a qualquer versão de código já existente. Observa-se também que, sempre que possível, as alterações realizadas em um mesmo arquivo são mescladas de maneira automática pela solução VCS (VCS – *Version Control System*).

O quadro 13, abaixo, sintetiza as vantagens na adoção de um sistema de controle de versões.

Quadro 13 – Vantagens proporcionadas por um sistema de controle de versões

Vantagens proporcionadas por um sistema de controle de versões
• Controle de histórico (máquina do tempo): é possível resgatar/acessar implementações anteriores para posterior consulta, revisão, restauração em caso de equívocos etc.; • Facilita o trabalho em equipe; • Permite a ramificação e a junção dos arquivos de um projeto (*branch / merge*) com maior facilidade; • Segurança (um determinado grupo de colaboradores, por exemplo, pode ter acesso somente a uma ramificação); • Organização (economia de espaço e trabalho).

Fonte: desenvolvido pelos autores (2021)

Independentemente do sistema de controle de versão utilizado, alguns termos são comuns a soluções encontradas no mercado, o quadro 14 abaixo, elenca a relação de termos e características presentes em VCS (*Version Control System*).

Quadro 14 – Termos e características encontrados em Sistemas de Controle de Versões

Termo/Característica	Descrição
item de configuração	Representa cada um dos elementos de informação que são criados, ou que são necessários, durante o desenvolvimento de um produto de software. eles devem ser identificados de maneira única e sua evolução deve ser passível de rastreamento.
repositório	É o local de armazenamento de todas as versões dos arquivos.
versão	Representa o estado de um item de configuração que está sendo modificado. Toda versão deve possuir um identificador único, ou VID (*Version Identifier*).
revisão	É uma versão que resulta de correção de defeitos ou implementação de uma nova funcionalidade. As revisões evoluem sequencialmente.
ramo	Uma versão paralela ou alternativa. Os ramos não substituem as versões anteriores e são usados concorrentemente em configurações alternativas.
espaço de trabalho	É o espaço temporário para manter uma cópia local da versão a ser modificada. Ele isola as alterações feitas por um desenvolvedor de outras alterações paralelas, tornando essa versão privada.
***check out* (clone)**	É o ato de criar uma cópia de trabalho local do repositório.
update	É o ato de enviar as modificações contidas no repositório para a área de trabalho.
commit	É o ato de criar o artefato no repositório pela primeira vez ou criar uma nova versão do artefato quando este passar por uma modificação,
merge	É©Ã a mesclagem entre versõÃµes diferentes, objetivando gerar uma Ãºnica versãÃ£o que agregue todas as alteraçõÃ§Ãµes realizadas.
changeset	É uma coleção atômica de alterações realizadas nos arquivos do repositório.

Fonte: Adaptado de Freitas (2010, p.9)

2.3.5.1 Evolução

O primeiro sistema de controle de versões foi desenvolvido por Marc J. Rockind no laboratório *Bell Labs* em 1972 e se chamava SCSS (*Source Code Control System*).

Posteriormente, em 1982, surge o RCS (*Revision Control System*), concebido por Walter F. Tinchy, cuja principal contribuição para esta categoria de solução foi a implementação da técnica de armazenamento *intervaled deltas*, que consiste nos primeiros passos para as técnicas de junção de arquivos.

A primeira geração de sistemas de controle de versão, formada pelo SCCSS e RCS era caracterizada por não possuir suporte a rede e trabalhar com apenas um arquivo de cada vez.

A segunda geração de versionadores, teve como expoentes os sistemas CVS (*Concurrent Versions System*) e o *Subversion*.

O sistema CVS (*Concurrent Versions System*), por sua vez apresentado em 1990, era uma evolução do RCS e tinha como virtude a possibilidade de abranger o gerenciamento de um projeto inteiro e não apenas um arquivo, por vez. como seu antecessor. Ele se tornou mais popular embora com alguns problemas de consistência e velocidade. A solução apresentada

neste período ainda não permitia renomear e mover arquivos, um eventual registro ou atualização de arquivos no repositório levava de 3 a 4 minutos. Em seguida, por volta de 2000, surge o *Subversion*, uma evolução do CVS, desenvolvido pela empresa CollabNet e observa-se nesta solução maior estabilidade e um tempo menor para comunicação com o repositório.

Moura (2013, p.20) afirma que:

> Uma das limitações dos versionadores de segunda geração é seu modelo centralizado. A maioria das operações necessitam de acesso a um servidor central. A mudança desse paradigma permitiu a criação dos versionadores de terceira geração, cuja principal característica é adotar um modelo distribuído e descentralizado.

A terceira e atual geração de sistemas de controle de versionamento é composta por soluções como *Git*, *BitKeeper*, *Mercurial*, *Bazaar* entre outras. Essa geração é responsável caracteriza-se por melhorar a velocidade de registro de atualizações e ter fluxos de trabalhos distribuídos.

A figura 8, apresentada a seguir, demonstra as soluções proprietárias e livres, subdivididas de acordo com seus respectivos modelos: centralizado ou distribuído.

Figura 8 – Sistemas de Controle de Versões

Fonte: Freitas (2010, p.8)

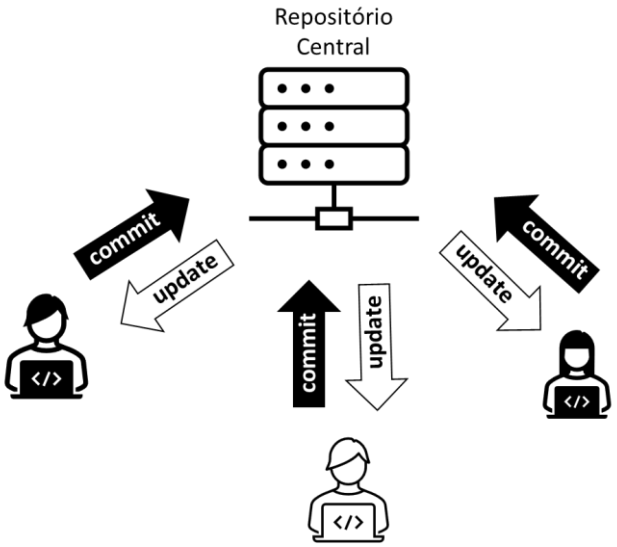
Um sistema de versionamento cujo modelo de operação é centralizado é caracterizado por um repositório central. Para que os *commits* e *updates* ocorram é necessário uma conexão constante com o servidor.

O controle de versão centralizado segue a topologia em estrela, havendo apenas um único repositório central mas várias cópias de trabalho, uma para cada desenvolvedor. A

comunicação entre uma área de trabalho e outra passa obrigatoriamente pelo repositório central (DIAS, 2016).

A figura 9, demonstra a esquematização do modelo centralizado

Figura 9 – Sistema de Controle de Versão com modelo centralizado



Fonte: desenvolvido pelos autores (2021)

Sistemas de controle de versões que operam com o modelo distribuído, como por exemplo o Git (terceira geração), caracterizam-se por contar com um repositório local, na máquina do desenvolvedor, e este por sua vez se conectará com um servidor remoto. Portanto ao realizar um *commit*, o desenvolvedor registrar as alterações no repositório local, que por sua vez poderá ser replicado ao repositório remoto pelo comando *push.* Para obter atualizações/revisões do repositório, o desenvolvedor deverá utilizar o comando *pull.*

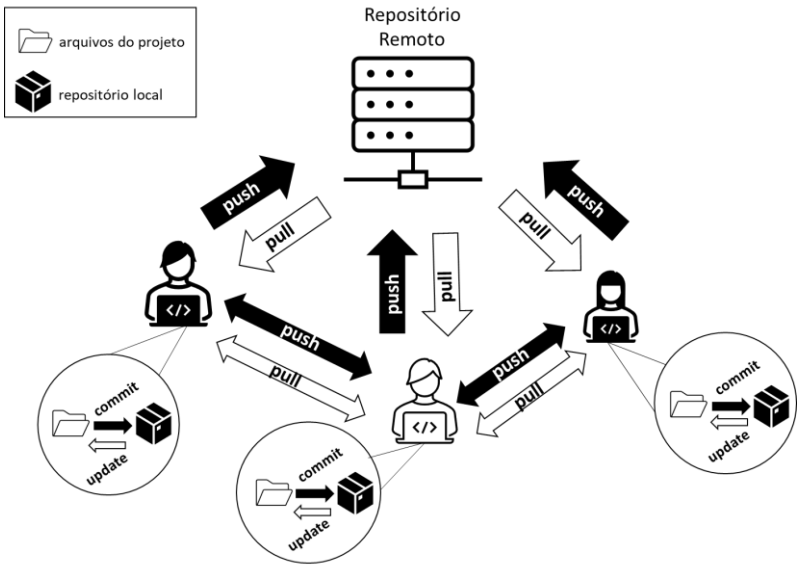
Ramos e Freitas (2010, p.3), afirmam que:

> No modelo distribuído, cada desenvolvedor trabalha diretamente em uma cópia local de arquivo e a atualização da cópia no repositório compartilhado é feita ao final de todas as modificações desejadas.

Um repositório recebe e envia revisões com qualquer outro através de operações *pull* e *push* sem a necessidade de uma topologia pré-definida. A sincronização entre os desenvolvedores acontece de repositório a repositório e não existe, em princípio, um repositório mais importante que o outro, embora o papel de um repositório central possa ser usado para convencionar o fluxo de trabalho. (DIAS, 2016)

A figura 10, demonstra a esquematização do modelo distribuído, presente em versionadores de terceira geração.

Figura 10 – Sistema de Controle de Versão com modelo centralizado



Fonte: desenvolvido pelos autores (2021)

2.3.5.2 O *Git* e o *GitHub*

No contexto de versionamento distribuído o *Git* é sistema de controle de versões livre, que conta com um repositório local e gerencia códigos fonte. A solução foi lançad~~a~~o em 2005, por Linus Torvalds, após o fim da relação entre a comunidade mantenedora do Kernel Linux e o *BitKeeper*, um sistema de controle de versão distribuído e não gratuito.

A revogação do inseto de pagamento e o rompimento da relação com a *BitKeeper*, levou Linus Torvalds a desenvolver seu próprio controle de versão, baseado nas experiências de uso da BetKeeper, visando obter uma melhor performance dentre todos os sistemas de controle de versão da época. (ALMEIDA, 2017)

A solução desenvolvida por Linus possui como base os seguintes pilares: velocidade, distribuído, flexibilidade para lidar com projetos grandes dimensões, robustez em relação ao suporte e desenvolvimento não-linear.

O Git trabalha com ramificações, ou seja, a cada commit registrando uma alteração de código, ele cria um novo ponto na ramificação atual (branch).

Como o trabalho concorrente é fundamental para o desenvolvimento de *software* em grande escala, vários projetos adotaram o uso do Git como seu sistema de controle de versões pela sua rapidez e eficiência. Projetos de desenvolvimento de software costumam possuir um repositório central tanto para administrar os versionamentos do produto como também a inclusão de novos requisitos e melhorias. Os desenvolvedores envolvidos ativamente na produção do software costumam possuir uma cópia local do repositório em seu ambiente de trabalho para realizar suas atividades de maneira independente. O resultado é um projeto sendo alterado continuamente em diversos aspectos por vários colaboradores ao mesmo tempo. (CUNHA, 2018, p.11)

O quadro 15, a seguir, apresenta um fluxo de contribuição em um projeto *Git*.

Quadro 15 – Fluxo de contribuição em um projeto Git

Etapas do fluxo	**Descrição**
1. Clone do Projeto	O primeiro passo para obter o código-fonte do projeto é cloná-lo em nossas máquinas, para que seus arquivos fiquem disponíveis localmente.
2. Criação da *Branch*	Ao criar uma *Branch*, estamos criando uma ramificação, totalmente independente, para podermos alterar os arquivos do projeto sem interferir nos originais. Esse processo é considerado uma boa prática quando se está trabalhando em nova funcionalidade.
3. *Commits*	Conforme vão sendo criados e alterados os arquivos, elas vão sendo divididas em *commits*. É importante que a descrição de cada *commit* seja objetiva, pois ela vai ficar salva no histórico das alterações.
4. *Push*	Uma vez que a funcionalidade está totalmente finalizada, devemos enviar nossa *Branch*, com todas as alterações, de volta ao repositório remoto. Assim, ela ficará disponível para os demais contribuidores do projeto poderem ver e alterar.
5. *Merge*	Para mesclar as modificações de sua *Branch* com os arquivos originais do projeto da *Branch* principal ou máster, você pode utilizar o comando Merge. Após isso, é necessário dar um commit e um push, para enviar a ramificação máster mesclada ao repositório remoto e deixar tudo disponível para os demais contribuidores. Existe também o *Pull Request*, que geralmente tem relação com a contribuição em projetos *open source.* Basicamente, ele ocorre quando se pede para o dono do repositório que suas modificações sejam incluídas nele.

Fonte: Adaptado de Hostgator(2020)

O *GitHub* por sua vez, foi criado em 2008 e consiste em uma plataforma, proprietária, atualmente da Microsoft, para criação de repositórios *Git.*

Segundo Kfouri *apud* Kalliamvakou (2019, p.9), o *GitHub* é um site que hospeda códigos de uma maneira colaborativa. Ele já possui mais de 10 milhões de repositórios e está se tornando uma das fontes de maior importância no quesito de artefatos de *software* na internet.

A solução provê funcionalidades interessantes como:

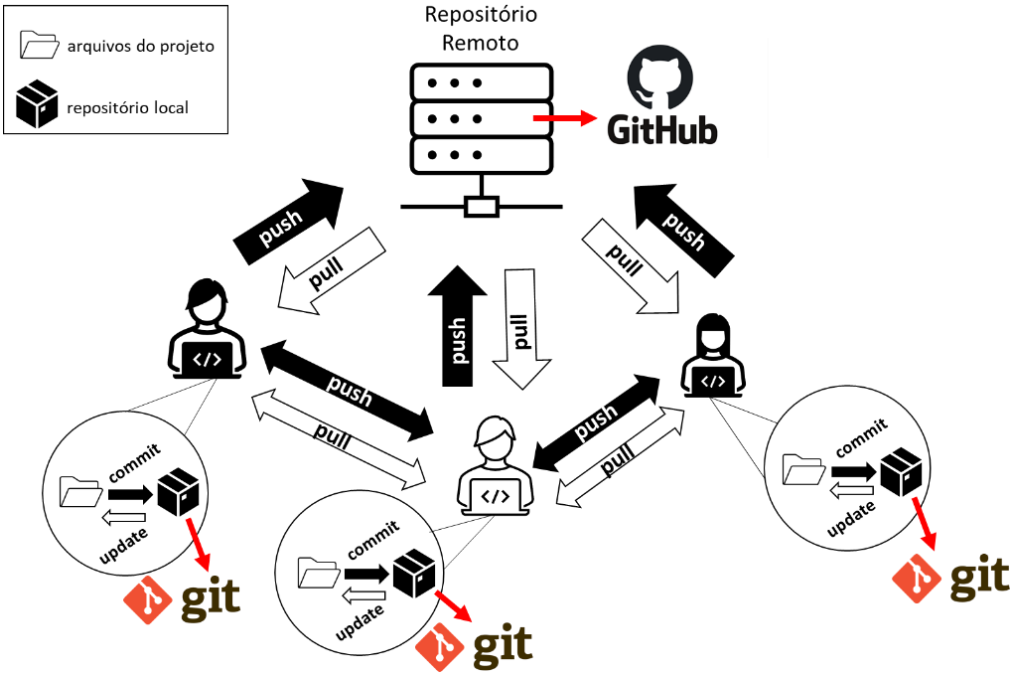
- Hospedagem de código de fonte de qualquer linguagem;
- Possibilidade de compartilhar, testar e colaborar com códigos;
- Desenvolvimento de networking (a plataforma conta com recursos de redes sociais);
- Forks (conceber um novo projeto com base em um existente, considerando as regras de licença do desenvolvedor/proprietário);
- Hospedagem simples (educacional).

Santos *et al*. (2017, p.3) apresentam as ações que um usuário pode realizar na plataforma:

> (i) *Follow:* indicando que um usuário segue outro usuário; (ii) conceder *stars* (estrelas) a projetos: quando um usuário acha que o projeto é interessante e tem qualidade e (iii) *watching: watcher* é o termo utilizado para descrever usuários que "seguem" determinados repositórios, recebendo atualizações e relatórios sobre a evolução do software. A quantidade de *"watchers"* serve como indicador da quantidade de atenção dada a um determinado projeto pela comunidade de desenvolvedores (Bissyandé et al., 2013). Os conceitos descritos abaixo estão relacionados especificamente à esta plataforma, são eles: (i) - *Commit:* Funcionalidade que permite efetuar o controle das alterações realizadas. É como se fosse um *"checkpoint"* do projeto. Sempre que for necessário é possível retroceder um *commit;* (ii) - *Branch:* É uma ramificação do seu projeto. Cada branch representa uma versão do seu projeto. Podemos seguir uma linha de desenvolvimento a partir de cada *branch;* (iii) - *Fork:* Consiste em realizar a cópia de um repositório que pertence a outra pessoa, adicionando esse repositório ao nosso repositório. Ou seja, nos tornamos os "proprietários" do repositório o qual estamos realizando o *fork*, porém o original se mantém intacto; (iv) – *Pull Request*: Consiste em uma solicitação de integração das nossas modificações com o repositório remoto. Basta informar na caixa de comentários um detalhamento do que foi criado/modificado. O responsável pelo repositório pode aceitar uma sugestão de mudança, ou até mesmo negar.

A figura 16, apresentada a seguir destaca a diferença entre *Git* e *GitHub* e a conexão existente entre os dois recursos. O *Git* consiste em um sistema de controle de versão distribuído e o *GitHub*, permite o desenvolvimento de repositórios remotos.

Figura 11 – Diferença entre *Git* e *GitHub*



Fonte: desenvolvido pelos autores (2021)

2.3.6 Framework Django

2.3.6.1 Visão Geral

O Django foi lançado em julho de 2005, nos EUA, pelos programadores Adrian Holovaty e Simon Willison que, na época, trabalhavam em um jornal local e precisavam criar uma solução que permitisse que o site deste jornal pudesse ser atualizado com mais frequência e de forma rápida.

O Django é um *framework web* de alto nível, escrito em linguagem de programação Python, de código aberto e gratuito, para o desenvolvimento ágil de aplicações *web*.

Dentre suas principais características estão a possibilidade de criar formulários de maneira automática, possuir um sistema de autenticação de usuários, assim como um um sistema de armazenamento de informações em cache que permite o registro de páginas dinâmicas (*caching*), permitir a serialização de objetos, além de já ter sido desenvolvido com proteção contra os mais comuns ataques na *web* (*Cross Site Scripting* (XSS), *Cross site request forgery* (CSRF), *SQL injection*, *Clickjacking*). O Django é um *framework* de acoplamento fraco

e coesão forte, o que significa que suas camadas atuam como módulos independentes, mas possuem alta sinergia entre si.

Um dos pontos fortes do *framework* Django é usar o mínimo possível de código para priorizar a agilidade no desenvolvimento de aplicações web e evitar redundâncias, conceito conhecido como *DRY*.

"*DRY* (*"Don't Reppeat Yourself"* [Não se repita ou Não fique se repetindo, em tradução livre]): É comum, no desenvolvimento para a web, ocorrer redundância de códigos entre as diversas camadas da aplicação; interface com o usuário em HTML; regras de negócio em uma ou (geralmente) várias classes no servidor; código que acessa e atualiza informações no seu SGBD; scripts de geração das tabelas; uma API REST para comunicação com outras interfaces com o usuário; e, muitas vezes, cada camada dessas possui códigos quase idênticos, resultando em um monte de ciclos de "copiar e colar", o que não é reuso, e sim redundância." (MACIEL, 2020, p. 280)

2.3.6.2 Padrão MTV: Model, Template , View

O framework Django adota o padrão de projeto MTV (*Model, Template, View*), uma variação do padrão MVC (M*odel, View, Controller*). Cada uma dessas camadas será explicada a seguir

2.3.6.2.1 Model

É a parte da aplicação que se comunica diretamente com o SGBD (Sistema Gerenciador de Banco de Dados) por meio do mapeamento objeto-relacional do Django.

Um dos maiores problemas para criação de aplicações web nos últimos anos, era como interagir de maneira transparente com o SGBD: escrever esse tipo de código é extremamente complexo e propenso a erros, sendo que existem várias ferramentas que simplificam essa tarefa fornecendo uma interface mais simples para as operações necessárias.

"Por exemplo: suponha que você crie uma tela com uma funcionalidade do tipo "Mestre-Detalhe", como um pedido e seus itens. Se fosse implementá-la sem um *software* de mapeamento OR, precisaria escrever código SQL para cada operação no banco de dados,

código Python para receber as informações das suas classes e mais código Python para converter uma representação em outra, afinal, espera-se que os programas Python usem um modelo de domínio orientado a objetos para representar seus dados, enquanto os SGBDs tipicamente usam um esquema relacional para armazená-los (daí o termo “objeto-relacional”). Você precisaria ainda controlar os dados para que, sempre que um registro mestre fosse visualizado, apagado ou editado, os registros filhos, na parte “detalhe” da tela, fossem modificados de acordo.” (MACIEL, 2020, p. 283)

Para solucionar esses problemas, o Django possui o seu próprio mapeamento OR, chamado de Django ORM (*Object-Relational Mapping*). O framework pode, então, criar as tabelas no banco de dados para você, gerenciar operações DML (*Data Manipulation Language*) como inclusão, alteração, exclusão e consulta de dados, além de controlar relacionamentos entre as tabelas que contêm informações. O Django suporta nativamente os SGBDs: PostgreSQL, MySQL, SQLite e Oracle. Além desses, existem também drivers fornecidos por terceiros, que permitem utilizar o *SQL Server* (Microsoft), DB2 (IBM), *SQL Anywhere* (SAP), *Firebird* e vários outros por meio de conexões ODBC.

2.3.6.2.2 Template

É a parte que o usuário visualiza, a camada de apresentação do sistema escrito com o Django. Os *templates* Django controlam a lógica de apresentação dos seus dados, ou seja, a forma como eles serão visualizados pelo usuário. Embora você possa criar seus *templates* com várias tecnologias diferentes, a maneira mais comum de implementar essa camada de software é por meio de páginas HTML decoradas com folhas de estilo em cascata (CSS — *Cascading Style Sheets*).

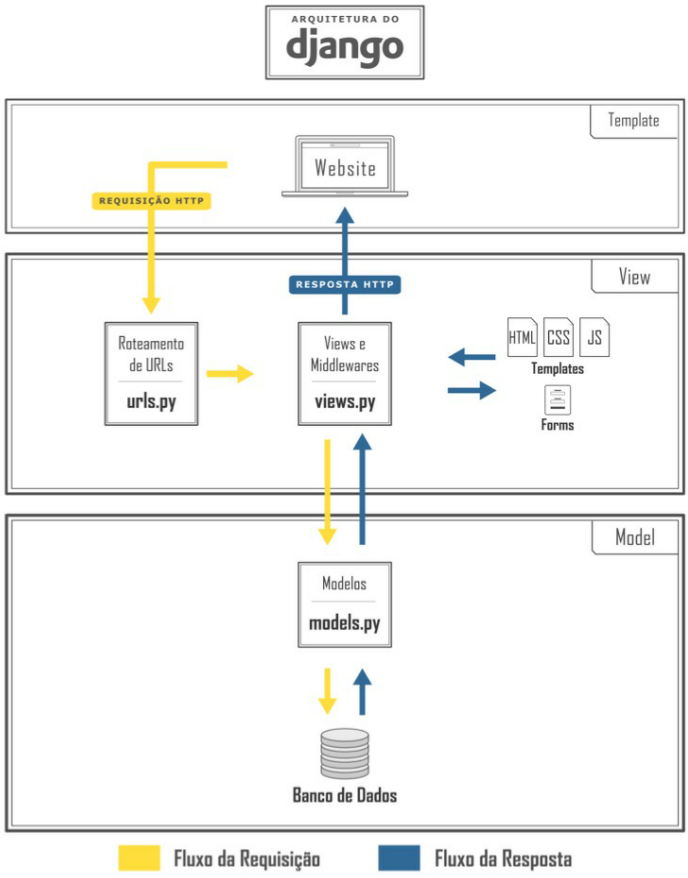
Os *templates* isolam a lógica de negócios da apresentação, permitindo que, se necessário, um designer trabalhe na criação da identidade visual do sistema enquanto um programador implementa as funcionalidades separadamente. Além disso, não é permitido executar código Python em *templates*, o que evita brechas de segurança. De fato, existe toda uma linguagem de *templates* embutida no Django, denominada DTL (*Django Template Language*), uma linguagem de script que usa *tags* para formatar o conteúdo a ser exibido. (MACIEL, 2020)

2.3.6.2.3 View

A *View* corresponde, no Django, à camada que controla a interação entre os dados (Modelo) e a apresentação (*Template)*. Ela embute as regras de negócio da sua aplicação. O nome escolhido para essa camada pode confundir aqueles que usam o padrão MVC (*Model-View-Controller*), para os quais o componente que realiza essa tarefa é denominado de Controlador (*Controller*) e "*view*" é o nome dado ao que o Django denomina de *template*.

Uma *view* normalmente devolve informações a um *template* na forma de uma variável ou um dicionário, para que seja montada uma página HTML de resposta a uma requisição. Entretanto, com o crescente uso de APIs REST, de uns anos para cá, também se tornou comum que as visões devolvam dados em formato JSON. (MACIEL, 2020)

Figura 12 – Arquitetura do Framework Django



Fonte: Python Academy (2020)

2.3.6.3 Principais Características

2.3.6.3.1 Formulários

O Django fornece uma gama de ferramentas e bibliotecas para auxiliar a construção de formulários para entrada de dados vindos dos seus visitantes, e então os processa e responde àquela entrada. Em HTML, um formulário é uma coleção de elementos dentro do comando <form>...</form> que permitem que os visitantes façam coisas como escrever um texto, selecionar opções, manipular objetos ou controle e assim por diante, e então enviar esta informação de volta ao servidor.

O formulário de autenticação do admin do Django contém vários elementos do tipo <*input*>: um *type*="*text*" para o nome do usuário, um de *type*="*password*" para a senha, e um *type*="*submit*" par ao botão "*Log in*". Ele também contém alguns objetos escondidos do tipo texto que o usuário não vê, o qual o Django usa para determinar o que fazer depois. Ele também diz ao navegador web que os dados do formulário devem ser enviados para a URL especificada no atributo <a*ction*> do <*form*> - /admin/ - e isso deve ser enviado usando o mecanismo HTTP especificado pelo atributo *method - post*.

No coração deste sistema de componentes está a classe Form do Django. Mais ou menos como o modelo do Django descreve a estrutura lógica de um objeto, seu comportamento, e a maneira como suas partes são apresentadas para nós, uma classe *Form* descreve um formulário e determina como este funciona e aparece. Os campos dos formulários são eles próprios classes; eles manipulam dados do formulário e fazem validações quando o formulário é enviado. Um campo de formulário, para o usuário, é representado como um *"widget"*. Cada campo tem uma classe de *Widget* padrão, mas isso pode ser sobrescrito quando requerido. (DJANGO SOFTWARE FOUNDATION, 2021)

2.3.6.3.2 Interface de Administração e Autenticação de Usuários e Permissões

Uma das partes mais poderosas do Django é a *interface* de administração automática. Ela lê metadados de seus modelos para fornecer uma *interface* rápida, onde usuários confiáveis podem gerenciar o conteúdo de seu site.

O Django também vem com um sistema de autenticação de usuário responsável por manipular contas de usuários, grupos, permissões e sessões baseados em *cookies.* O sistema de autenticação consiste em: Usuários; Permissões: *flags* binário (*yes/no*) designando quando um usuário pode executar uma certa tarefa; Grupos: uma forma genérica de aplicar *labels* e permissões para mais de um usuário; e Mensagens: uma forma simples de enfileirar mensagens para um dado usuário. (DJANGO SOFTWARE FOUNDATION, 2021)

2.3.6.3.3 Caching

Uma dilema essencial dos sites dinâmicos vem a ser o próprio fato de serem dinâmicos. Cada vez que um usuário requisita uma página, o servidor *web* faz todo o tipo de cálculos – consultas a bancos de dados, renderização de *templates* e lógica de negócio – para criar a página que o seu visitante vê. Isso tem um custo de processamento muito maior que apenas a leitura de arquivos estáticos no disco.

Para a maior parte dos aplicativos web, esse *overhead*[2] não é um problema. A maior parte das aplicações web são de sites pequenos com tráfego equivalente. Mas para aplicações de porte médio para grande, é essencial eliminar toda a sobrecarga possível.

É onde entra o cache. Fazer o *cache* de algo é gravar o resultado de um cálculo custoso para que você não tenha de executar o cálculo da próxima vez. Aqui está um pseudocódigo explicando como isso funcionaria para uma página *web* gerada dinamicamente:

Figura 13 – Exemplo de Caching

```
tente encontrar uma página no cache para tal URL
se a página estiver no cache:
    retorne a página do cache
se não:
    gere a página
    guarde a página gerada no cache (para a próxima vez)
    retorne a página gerada
```

Fonte: DJANGO SOFTWARE FOUNDATION (2021)

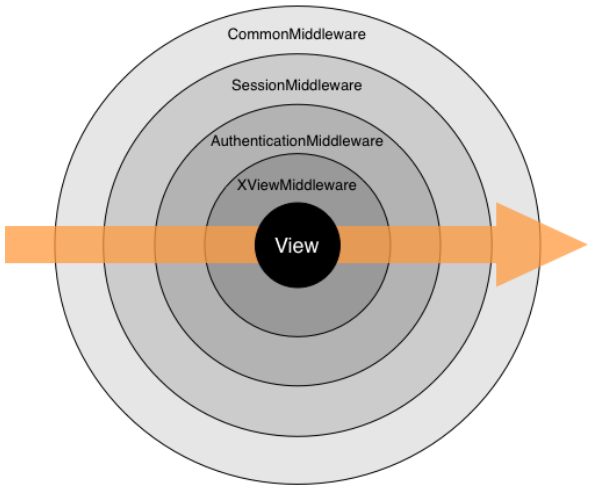
[2] Overhead, em ciência da computação, é geralmente considerado qualquer processamento ou armazenamento em excesso, seja de tempo de computação, de memória, de largura de banda ou qualquer outro recurso que seja requerido para ser utilizado ou gasto para executar uma determinada tarefa.

O Django vem com um sistema de cache robusto que permite que você guarde as páginas dinâmicas para que elas não tenham de ser calculadas a cada requisição. Por conveniência, o Django oferece diferentes níveis de granularidade de cache: Pode-se fazer o *cache* da saída de *views* específicas, ou somente o cache das partes que são difíceis de produzir, ou ainda, o *cache* do site inteiro. (DJANGO SOFTWARE FOUNDATION, 2021)

2.3.6.3.4 Middleware

O *Middleware* é um *framework* de *hooking*[3] dentro do processamento de requisição/resposta do Django. Ele é um sistema de *"plugins"* leve, e de baixo nível para alterar globalmente a entrada ou saída do Django. Cada componente *middleware* é responsável por fazer alguma função específica. Por exemplo, o Django inclui um componente *middleware* chamada *XViewMiddleware*, que adiciona um cabeçalho HTTP "*X-View*" a toda resposta para uma requisição *HEAD* (Tipo de Método HTTP). (DJANGO SOFTWARE FOUNDATION, 2021)

Figura 14 – Exemplos de *Middlewares* na Camada *View*



Fonte: DJANGO SOFTWARE FOUNDATION (2021)

[3] Hooking, do inglês enganchar, cobre uma série de técnicas utilizadas para modificar ou melhorar o comportamento de um sistema operacional, aplicações ou outros componentes de software através da interceptação de chamadas de funções, mensagens ou eventos passados entre componentes de software.

2.3.6.3.5 Serialização de Dados

O *framework* de serialização do Django provê um mecanismo para "traduzir" objetos do Django em outros formatos. Usualmente estes outros formatos serão baseados em texto e usados para enviar objetos do Django por um *Thread*, mas é possível para um serializador manipular qualquer formato (baseado em texto ou não). (DJANGO SOFTWARE FOUNDATION, 2021)

2.3.6.3.6 Segurança

Como dito anteriormente, o Django já possui funcionalidades que garantem proteção contra os ataques mais comuns na web. A seguir uma breve descrição sobre cada um deles:

Cross Site Scripting (XSS): Ataques XSS permitem a um usuário injetar scripts dentro do browser de outros usuários. Isso geralmente é feito armazenando scripts maliciosos no banco de dados onde ele será obtido e exibido para outros usuários, ou fazendo usuários clicarem em um link que faz com que o *JavaScript* do invasor seja executado pelo *browser* do usuário. Entretanto, ataques XSS podem se originar de qualquer fonte de dados não confiável, tais como cookies ou serviços web, sempre que os dados não são suficientemente limpos antes da inclusão em uma página. Usar *templates* Django protege contra a maioria dos ataques XSS.

Cross site request forgery (CSRF): Ataques de solicitações forjadas entre sites, na sigla em inglês, CSRF, permitem que um usuário malicioso execute ações usando as credenciais de outro usuário sem o seu conhecimento. O Django já possui proteção embutida contra a maioria dos tipos de ataques CSRF, contanto que você a tenha habilitado.

SQL injection: É um tipo de ataque onde o usuário malicioso consegue executar código SQL arbitrário em um banco de dados. Isso pode resultar em registros sendo deletados ou vazamento de dados. Os *querysets* do Django são protegidos contra injeção de SQL, uma vez que suas consultas são construídas usando parametrização de consulta. O código SQL de uma consulta é definido separadamente dos parâmetros da consulta. Como os parâmetros podem ser fornecidos pelo usuário e, portanto, não seguros, eles são filtrados pelo *driver* de banco de dados subjacente.

Clickjacking: Roubo de *click*, é um tipo de ataque onde um site malicioso embrulha outro site dentro de um frame. Esse ataque pode resultar em um usuário desavisado sendo levado a fazer ações não intencionadas no site alvo. O Django possui proteção contra *clickjacking* no form do *middleware X-Frame-Options* que em um *browser* com suporte pode prevenir um site de ser renderizado dentro de um *frame*. (DJANGO SOFTWARE FOUNDATION, 2021)

2.3.6.3.7 Internacionalização

O Django tem um suporte completo para internacionalização dos textos dos códigos e *templates*. O objetivo da internacionalização é permitir que uma única aplicação *web* ofereça seus conteúdos e funcionalidades em múltiplas linguagens. O Django permite que o desenvolvedor ou autor de *templates* especifique quais partes de sua aplicação podem ser traduzidas, e usa *hooks* para traduzir a aplicação web para um usuário em particular de acordo com sua preferência. (DJANGO SOFTWARE FOUNDATION, 2021)

2.4. Metodologia

O presente projeto está fundamentado em pesquisa bibliográfica e prototipação/desenvolvimento da solução proposta. Essa pesquisa será desenvolvida em sete etapas:

1. identificação e seleção do material bibliográfico pertinente ao tema;
2. leitura e fichamento em formato digital do material selecionado com identificação de obas, autores e ideias centrais;
3. entendimento do cenário a ser explorado (contexto atual e perspectivas mediante construção da solução), por meio de pesquisas, entrevistas, mapeamento de processos e outros;
4. identificação de requisitos mínimos para desenvolvimento da solução;
5. prototipação da solução, com base na demanda existente e com o objetivo de fornecer subsídios para construção do sistema.
6. Testes com os potenciais usuários (docentes, coordenadores e demais membros da área pedagógica da instituição), para validação e possíveis aperfeiçoamentos;

7. Registro de resultados obtidos e análise dos procedimentos realizados.

Em relação ao desenvolvimento da solução, além de empregar as tecnologias necessárias para concepção da ferramenta e consequente resolução do problema, o grupo optou por utilizar as técnicas de *Design Thinking*, que segundo Arrudas (2020), é uma metodologia de desenvolvimento de produtos e serviços focados nas necessidades, desejos e limitações dos usuários, cujo grande objetivo é converter dificuldades e limitações em benefícios para o cliente e valor de negócio para a sua empresa.

O quadro 16, a seguir, sintetiza as etapas do *Design Thinking*, apresentadas na percepção de alguns autores por Reche e Muniz (2018, p.6)

Quadro 16 – Etapas do *Design Thinking*

Etapa	**Descrição**
Pensamento	Também conhecida como a etapa da descoberta ou imersão, nela se busca o entendimento sobre o tema ou problema a ser solucionado. Há a necessidade de um aprofundamento no objetivo e no ponto de partida do processo. Através das informações do cliente, é possível identificar restrições mentais que proporcionam um referencial de por onde iniciar o projeto, além de compreender os objetivos a serem atingidos. Conforme a evolução do projeto, as restrições do problema por diversas vezes são desafiadas ou até mesmo mudadas pela equipe ou cliente. É importante imergir e compreender os limites envolvidos no projeto desde seu início.
Pesquisa	Na etapa de pesquisa, há a coleta de informações e seu uso para aprofundamento do tema, por isso ela também pode ser chamada de etapa de interpretação. Se os consumidores não sabem expressar diretamente o que gostam ou odeiam, é necessário pesquisar como vivenciam a experiência da marca, produto ou serviços, para poder identificar oportunidades implícitas ou disfarçadas. Não é fácil decidir quais técnicas de pesquisa utilizar, quem observar, quando parar e iniciar a síntese das informações. Tudo requer prática e o período de observação pode ser longo, mas, seguindo a premissa de primar pela qualidade e não pela quantidade, no final serão coletados dados importantes para o projeto. Assim que os dados são coletados, é necessário analisá-los para identificar padrões, cujo papel é essencial no processo para a criação de opções e escolhas pela análise e síntese das informações encontradas.
Ideação	É a etapa de criação, em que são geradas as ideias e soluções relacionadas à questão abordada pelo *Design Thinking*. Após o entendimento do tema e a coleta das informações necessárias para entendê-lo, os designers possuem os insumos para gerar insights e soluções pertinentes ao problema. Os insights são estímulos ou pequenas partes de uma informação maior coletada por um indivíduo. Quando há um conjunto de insights, é possível montar cenários, compreender relações, hábitos e crenças. Não existe o insight perfeito, mas o que o torna importante é o que pode ser feito com ele. Esta etapa permite uma cocriação na busca de novas soluções. Por isso, se faz necessário que ela seja participativa e que todos os envolvidos tenham conhecimento dos objetivos do projeto para que sua implementação seja mais assertiva e possa alcançar os melhores resultados. A criação de ideias no processo de *Design Thinking* tem uma transição recorrente entre os estágios de divergência e convergência. Na fase divergente os designers levantam e discutem novas opções para o problema e na fase convergente o oposto, eliminam opções e fazem suas escolhas dentre as ideias sugeridas. Na revisão desta etapa, é necessário haver um momento de selecionar as melhores ideias e as mais promissoras para testá-las.

Experimentação	Na etapa de experimentação, são feitos os testes, prototipagens das opções escolhidas. Baseia-se em tirar as ideias do papel para testar se realmente são boas alternativas. A prototipagem é importante, pois gera resultado com mais rapidez. Apesar de tomar certo tempo, é a melhor forma de escolher entre vários direcionamentos possíveis que uma ideia pode gerar. Quanto antes se tonarem tangíveis as ideias, mais rápido será possível avaliá-las, aprimorá-las e apontar melhores alternativas. Na prototipação, procura-se descobrir as ideias que funcionam ou que podem ser trabalhadas para dar continuidade. Nesta etapa são eliminados os protótipos de *insights* que não tenham potenciais. O desenvolvimento de protótipos deve ser barato e rápido, pois permite a exploração de ideias paralelas que deverão passar por prototipagem e necessitam de investimento. A prototipagem resulta no aprendizado do que se deve ou não fazer, dos melhores meios para chegar ao resultado almejado e identificar de forma antecipada as barreiras que possam prejudicar a adoção da nova estratégia escolhida.
Desenvolvimento	Nesta etapa, há a implementação das novas ideias desenvolvidas no processo de *Design Thinking*. Após os protótipos serem testados incansavelmente na etapa de implementação, a equipe concentra-se em disseminar a ideia de forma clara para ser aceita por toda a empresa, visando demonstrar e comprovar sua funcionalidade no objetivo estratégico da organização. O momento de desenvolvimento configura-se na entrega da solução do projeto de design. A equipe envolvida no projeto foca a atenção nesta etapa para garantir que os resultados possam atingir as expectativas de todos os envolvidos. O maior esforço despendido é para manter o projeto dentro do prazo necessário para que nenhuma das variáveis sofra algum tipo de alteração que comprometa o projeto. Esta é a fase mais longa do projeto e também a mais desafiadora tecnicamente.
Evolução	A etapa de evolução desenvolve-se pelo feedback da implementação. Nela as ideias geradas pelo *Design Thinking* podem evoluir por meio de mudanças e afinamentos com foco em melhorias e novas alternativas. A fase de *feedback* é o momento em que todo o projeto é avaliado com o objetivo de identificar os aspectos positivos e os que precisam ser melhorados após o desenvolvimento que a solução implementada foi bem recebida, quais seus efeitos sobre o público-alvo, como foi a resposta sobre a solução proposta e qual foi o aprendizado para projetos futuros. Por mais que pareça ser a última etapa do processo de *Design Thinking*, a evolução ocorre ao longo de todo o projeto, pois a cada etapa há a possibilidade de aprender algo novo que possa abrir novos caminhos e alternativas para o sucesso do projeto. É importante que em cada etapa seja observado onde se está, para onde se quer ir, o que está funcionando bem e o que não está. Mesmo que muita coisa esteja errada, não se pode parar de evoluir o projeto até que dê certo.

Fonte: Reche e Muniz (2018, p.6)

REFERÊNCIAS

ALMEIDA, Eduardo**. Breve história do Git, o controle de versão mais adotado pelos desenvolvedores do mundo inteiro**. [S. l.], 7 dez. 2017. Disponível em: https://medium.com/@dyhalmeida/breve-hist%C3%B3ria-do-git-o-controle-de-vers%C3%A3o-mais-adotado-pelos-desenvolvedores-do-mundo-inteiro-c82466b50c02. Acesso em: 5 out. 2021.

ALVES, W. P. **Projetos de Sistemas Web Conceitos, Estruturas, Criação de Banco de dados e Ferramentas de Desenvolvimento**. São Paulo: Editora Saraiva, 2019. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788536532462/. Acesso em: 03 Oct 2021

ANDRADE, Maria do Carmo Ferreira de. **Roteiro para elaboração de situação de aprendizagem. 2015. Dissertação de Mestrado. Instituto Federal do Amazonas**, [S. l.], 2015. Disponível em: http://repositorio.ifam.edu.br/jspui/handle/4321/401. Acesso em: 22 set. 2021.

ARRUDAS, Mariana. **O que significa Design Thinking?.** [S. l.], 5 mar. 2020. Disponível em: http://www.inovacao.usp.br/o-que-significa-design-thinking/. Acesso em: 5 out. 2021.

BALDISSERA, Renê.Luiz.dos. S. **Desenvolvimento de uma Ferramenta para Gestão de Carreira e Vendas voltada ao Mercado Audiovisual. Monografia - Curso de Engenharia de Controle e Automação, Universidade Federal de Santa Catarina**. Florianópolis. 2019.

BARBOZA, Fabrício.Felipe. M.; FREITAS, Pedro.Henrique. C. ***Modelagem e desenvolvimento de banco de dados*.** Grupo A, 2018. 9788595025172. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595025172/. Acesso em: 10 out. 2021.

CALIARI, F. M. **Método para construção de ontologias a partir de diagramas entidade--relacionamento. 2007. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) — Universidade Tecnológica Federal do Paraná**, Curitiba, 2007. Disponível em: https://bit.ly/2I9G0Yb. Acesso em: 19 mar. 2020.

CASEIRO, Cíntia Camargo Furquim; GEBRAN, Raimunda Abou. **AVALIAÇÃO FORMATIVA: CONCEPÇÃO, PRÁTICAS E DIFICULDADES**. Revista Nuances, [s. l.], ano XIV, n. 16, 2008.

CARVALHO, André C. P. L. F. de. Introdução à computação: hardware, software e dados / André C. P. L. F. de Carvalho, Ana Carolina Lorena. - 1. ed. - Rio de Janeiro: LTC, 2017.

COUGO, P. S. **Modelagem conceitual e projeto de banco de dados**. Rio de Janeiro: Elsevier, 1997.

CUNHA, Marcela Bandeira. **Entendendo o Uso do Git em Equipes de Desenvolvimento de Software**. 2018. Monografia (Graduação) (Engenharia da Computação) - Universidade Federal de Pernambuco, [S. l.], 2018. Disponível em: https://www.cin.ufpe.br/~tg/2018-2/TG_EC/tg-mbc3.pdf. Acesso em: 5 out. 2021.

DELORS, J. **Educação: um tesouro a descobrir**. 2ed. São Paulo: Cortez Brasília, DF: MEC/ UNESCO, 2003.

DIAS, André Felipe. **Conceitos Básicos de Controle de Versão de Software - Centralizado e Distribuído**. [S. l.], 11 maio 2016. Disponível em: https://blog.pronus.io/posts/controle-de-versao/conceitos-basicos-de-controle-de-versao-de-software-centralizado-e-distribuido/. Acesso em: 7 out. 2021.

DJANGO SOFTWARE FOUNDATION. **Documentação do Django**: Tudo o que você precisa saber sobre Django. [*S. l.*], 2021. Disponível em: https://docs.djangoproject.com/pt-br/3.2/. Acesso em: 11 out. 2021.

ESCOBAR, Herton. **Informação versus conhecimento: Tempo disponível para leitura e reflexão pode ter chegado ao limite**. Ciência - Estadão, [S. l.], p. 1-2, 12 fev. 2014. Disponível em: https://ciencia.estadao.com.br/blogs/herton-escobar/informacao-versus-conhecimento. Acesso em: 21 set. 2021.

FERREIRA, Ana Paula. **O Que é Indústria 4.0 na pratica e Como Ela Vai Impactar a indústria mundial e a brasileira por Genilson Pavão**. [S. l.], 11 maio 2018. Disponível em: http://www.engcomp.uema.br/?p=673. Acesso em: 21 set. 2021.
FILETO, R. **O Modelo Entidade-Relacionamento**. 2006. Disponível em: <www.inf.ufsc. br/~r.fileto/Disciplinas/INE5423-2010-1/Aulas/02-MER.pdf>. Acesso em: Acesso em: 10 out. 2021.

FREITAS, Daniel Tannure Menandro de. **Análise Comparativa entre Sistemas de Controle de Versões**. 2010. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal de Juiz de Fora, [S. l.], 2010. Disponível em: http://www.ufjf.br/getcomp/files/2013/03/An%C3%A1lise-Comparativa-entre-Sistemas-de-Controle-de-Vers%C3%B5es-Daniel-Tannure-Menandro-de-Freitas.pdf. Acesso em: 8 out. 2021.

FÜHR, Regina Candida. **Educação 4.0 e seus impactos no século XXI**. Editora Realize, [S. l.], p. 1-6, 6 ago. 2018. Disponível em: https://www.editorarealize.com.br/editora/anais/conedu/2018/TRABALHO_EV117_MD4_SA19_ID5295_31082018230201.pdf. Acesso em: 22 set. 2021.

GIL, Antônio Carlos. **Metodologia do Ensino Superior**. São Paulo: Atlas, 2020. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788597023954/. Acesso em: 24 set. 2021.

GIMENES, I. M. S.; HUZITA, E. H. M. **Desenvolvimento baseado em componentes: conceitos e técnicas**. São Paulo: Ciência Moderna, 2005.

HEUSER, C. A. **Projeto de banco de dados**. Porto Alegre: Bookman, 2009. (Série Livros Didáticos Informática UFRGS).

HOSTGATOR. **Conheça o Git, o sistema de versionamento que protege os projetos**. [S. l.], 1 jul. 2020. Disponível em: https://www.hostgator.com.br/blog/git-o-sistema-de-controle/. Acesso em: 6 out. 2021.

KFOURI, Tiago de Oliveira. **Análise de projetos no GitHub que utilizam Behavior Driven Development**. 2019. xi, 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)—Universidade de Brasília, Brasília, 2019. Disponível em: https://bdm.unb.br/handle/10483/24419. Acesso em: 7 out. 2021.

LJUBOMIR, PERKOVIC. I**ntrodução à Computação Usando Python - Um Foco no Desenvolvimento de Aplicações**. São Paulo: Grupo GEN, 2016. 9788521630937. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788521630937/. Acesso em: 03 out. 2021.

MACHADO, F. N. R. **Banco de dados: projeto e implementação.** São Paulo: Érica, 2014.
OLIVEIRA, C. H. P. SQL: curso prático. São Paulo: Novatec, 2002.
MACIEL, F.M.D. B. **Python e Django**. Rio de Janeiro: Editora Alta Books, 2020. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786555200973/. Acesso em: 03 Oct 2021.

MOURA, Marcello Henrique Dias de. **Comparação entre desenvolvedores de software a partir de dados obtidos em repositório de controle de versão**. 2013. 126 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Goiás, Goiânia, 2013. Disponível em: http://repositorio.bc.ufg.br/tede/handle/tede/7944. Acesso em: 9 out. 2021.

MANZANO, José.Augusto.N. G. ***MySQL 5.5 Interativo: Guia Essencial de Orientação e Desenvolvimento*****.** Editora Saraiva, 2011. 9788536519449. Disponível em:https://integrada.minhabiblioteca.com.br/#/books/9788536519449/. Acesso em: 10 out. 2021.MIAMOTO, C. V. P. **Refinamento de um Diagrama Entidade-Relacionamento: estudo de caso em um sistema ERP**. 2012. 24 f. Monografia (Curso de Especialização em Informática com ênfase em Análise Orientada à Objetos) - Universidade Federal do Paraná, Curitiba, 2012. Disponível em: <https://acervodigital.ufpr.br/bitstream/handle/1884/38542/R%20-%20E%20-20CRISTIANE%20VIEIRA%20PROENCA%20MIAMOTO.pdf?sequence=1&isAllowed=y>. Acesso em: 10 out. 2021.

OKUYAMA, Fabio Yoshimitsu. **Desenvolvimento de software I : Conceitos básicos** – Porto Alegre : Bookman, 2014.

PALESTINO, Caroline Munhoz Corrêa**. Estudo de tecnologias de controle de versões de software**. 2015. 72 f. Trabalho de Conclusão de Curso (Bacharel em Gestão da Informação) – Universidade Federal do Paraná, [S. l.], 2015. Disponível em: https://acervodigital.ufpr.br/handle/1884/41087. Acesso em: 9 out. 2021.

PASSOS, Marize Lyra Silva. **Da Educação 1.0 a Educação 4.0: os caminhos da educação e as novas possibilidades**. [S. l.], 31 ago. 2019. Disponível em: https://www.marizepassos.com/post/educa%C3%A7%C3%A3o-1-0-a-educa%C3%A7%C3%A3o-4-0-os-caminhos-da-educa%C3%A7%C3%A3o-e-as-novas-possibilidades-para-a-educa%C3%A7%C3%A3o.

PICHETTI, Roni. F.; VIDA, Edinilson.da. S.; CORTES, Vanessa.Stangherlin.Machado. P. **Banco de Dados**. Grupo A, 2021. 9786556900186. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786556900186/. Acesso em: 10 out. 2021.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software: uma abordagem profissional. 7**. ed. Porto Alegre: AMGH, 2011.
PROPOSTA Pedagógica Senac. [S. l.], 2005. Disponível em: https://www.sp.senac.br/pdf/53727.pdf. Acesso em: 20 set. 2021.

RAMOS, Erivan de S., FREITAS, Rejane C. **Análise Comparativa de Sistemas de Controle de Versões Baseados em Código Aberto**. Disponível em: http://www.infobrasil.inf.br/userfiles/28-05-S1-1-68581-Analise%20Comparativa.pdf. Acesso em: 4 out. 2021.

RECHE, Marcelo Mesquita; MUNIZ, Raquel Janissek**. Inteligência estratégica e design thinking: conceitos complementares, sequenciais e recorrentes para estratégia inovativa**. Rio de Grande do Sul, 2018. Disponível em: https://lume.ufrgs.br/handle/10183/187347. Acesso em: 4 out. 2021.

RODRIGUES, J**. Modelo Entidade Relacionamento (MER) e Diagrama Entidade-Relacionamento (DER).** 2014. Disponível em: <https://www.devmedia.com.br/modelo--entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332>. Acesso em: 10 out. 2021.

SANTOS, Maycon Medeiros de F; SPIES, João Henrique Lopes; WEITZEL, Leila. **Análise do GitHub como rede social e rede de colaboração**. International association for development of the information society, Algrave, Portugal, ed. 10, p. 135-142, 2017. Disponível em: http://www.iadisportal.org/digital-library/an%C3%A1lise-do-github-como-rede-social-e-rede-de-colabora%C3%A7%C3%A3o. Acesso em: 6 out. 2021.

SEBRAE. **A Quarta Revolução Industrial e o futuro do trabalho**. [S. l.], 2017. Disponível em: https://www.sebrae.com.br/sites/PortalSebrae/artigos/futuro-dos-trabalhos-voce-sabe-qual-e,900553c03a730610VgnVCM1000004c00210aRCRD. Acesso em: 21 set. 2021.

SENAC**. Modelo Pedagógico Nacional - Concepções e Princípios**. São Paulo, 2015. Disponível em: http://www.extranet.senac.br/modelopedagogicosenac/arquivos/DT_1_Concepcoes%20e%20Principios.pdf. Acesso em: 21 set. 2021.

SENAC. **Orientações para a prática pedagógica:** Jeito Senac de Educar. São Paulo: Senac, 2016a.

SENAC. **Orientações para a prática pedagógica:** Planejar. São Paulo: Senac, 2016b.

SENAI-RS.A **Indústria 4.0 chegou no Brasil?.** [S. l.], 2019. Disponível em: https://www.sesirs.org.br/industria-inteligente/industria-40-chegou-no-brasil. Acesso em: 20 set. 2021.

SILVA, Lidiane Pereira da; VIANA, Flávia Roldan. **Plano De Aula Colaborativo:** Uma Proposta No Contexto Da Educação Inclusiva. Revista Prometeu, v. 6, n. 1, 1 jan. 2021. Disponível em: https://periodicos.ufrn.br/revprometeu/article/view/25409. Acesso em: 24 de set. de 2021.

SOMMERVILLE, Ian. **Engenharia de Software.** São Paulo : Pearson Prentice Hall, 2019. Disponível em https://plataforma.bvirtual.com.br/Leitor/Publicacao/168127/pdf/490?code=sM2sK/ulOuQ7wV+8FoyRHLLIhG06LEnrNt2hp781VtCRBzi2Je9wEGbP0C9adNEiyvX3ALBaCkguWIImXbgM8Q==. Acesso em: 08 out. 2021.

SPÍNOLA, Rodrigo Oliveira. **Engenharia de Software Magazine Ano 1** – 1° Edição de 2007, Qualidade de Software – Entenda os principais conceitos sobre testes e inspeção de Software. Disponível em: <https://www.devmedia.com.br/revista-engenharia-de-software/8028>. Acesso em 10 Out 2021.

TAKAHASHI, Regina Toshie; FERNANDES, Maria de Fátima Prado. **Plano de Aula: conceitos e metodologia**. **Acta Paul Enferm**., v. 17, n. 1, p. 114-118, jan. 2004. Disponível em https://acta-ape.org/wp-content/uploads/articles_xml/1982-0194-ape-S0103-

210020040001700059 5/1982-0194-ape-S0103-2100200400017000595.pdf. Acesso em: 24 de set. 2021.

VETORAZZO, Adriana de Souza. **Engenharia de software** / Adriana de Souza Vettorazzo ; [revisão técnica : Fábio Josende Paz]. Porto Alegre : SAGAH, 2018.

WHITING, Kate. **These are the top 10 job skills of tomorrow – and how long it takes to learn them.** [S. l.], 21 out. 2020. Disponível em: https://www.weforum.org/agenda/2020/10/top-10-work-skills-of-tomorrow-how-long-it-takes-to-learn-them/. Acesso em: 20 set. 2021.

ZANIN, Aline. **Qualidade de software / Aline Zanin.** [et al.] ; [revisão técnica: Maria de Fátima Webber do Prado Lima]. – Porto Alegre: SAGAH, 2018.

ZENKER, A. M.; SANTOS, J. C. D.; COUTO, J.M. C.; AL., et. **Arquitetura de sistemas**. Porto Alegre: Grupo A, 2019. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595029767/. Acesso em: 08 out. 2021.