Universidade Estadual de Londrina

GUILHERME SPATI

FRAMEWORK DE AVALIAÇÃO DE MATURIDADE PARA PRÁTICAS DE GERENCIAMENTO E DESENVOLVIMENTO DE SOFTWARE BASEADO NO ITIL4

LONDRINA

2023

GUILHERME SPATI

FRAMEWORK DE AVALIAÇÃO DE MATURIDADE PARA PRÁTICAS DE GERENCIAMENTO E DESENVOLVIMENTO DE SOFTWARE BASEADO NO ITIL4

Versão Preliminar de Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof(a). Dr(a). Rodolfo Miranda de Barros

LONDRINA
2023

“The concept of waiting bewilders me.
There are always deadlines. There are
always ticking clocks. That is why you must
manage your time.“
(Whiterose, Mr. Robot)

SPATI, GUILHERME. **Framework de Avaliação de Maturidade para Práticas de Gerenciamento e Desenvolvimento de Software baseado no ITIL4**. 2023. 42f. Trabalho de Conclusão de Curso – Versão Preliminar (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

Atualmente, a maior parte das organizações depende dos serviços prestados pela área de Tecnologia da Informação, tornando esta um objeto de extrema importância para a garantia de sucesso da organização. Possuir um processo de desenvolvimento de *software* é imprescindível para que a qualidade do *software* seja o objetivo, gerando mais valor ao cliente, visto que a indústria nesta área esta em crescente expansão, tornando o mercado cada vez mais competitivo. Sendo assim, o presente trabalho propõe o desenvolvimento de um *framework* para diagnosticar o grau de maturidade da prática de gerenciamento e desenvolvimento de *software* de acordo com o ITIL4.

Palavras-chave: ITIL 4. Framework. Gerenciamento e Desenvolvimento. Software

SPATI, GUILHERME. **Maturity Assessment Framework for Software Management and Development Practices based on ITIL4**. 2023. 42p. Final Project – Draft Version (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2023.

ABSTRACT

Currently, the majority of organizations rely on services provided by the Information Tecnology department, making it an extremely important element for ensuring the organization's success. Having a software development process is essential to prioritize software quality, generating more value for the customer, especially as the industry in this field is experiencing continuous growth, leading to an increasingly competitive market. Therefore, this present work proposes the development of a framework to diagnose the maturity level of software management and development practices according to ITIL4.

Keywords: ITIL 4. Framework. Management and Development. Software.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

TI	Tecnologia da Informação
TIC	Tecnologia da Informação e Comunicação
UEL	Universidade Estadual de Londrina
CMMI	*Capability Maturity Model Integration*
ITIL	*Information Technology Infrastructure Library*
SVS	Sistema de Valor de Serviço
XP	*Extreme Programming*
ISO	*International Organization for Standardization*
BDUF	*Big Design Up Front*
MVP	*Minimum Viable Product*
PO	*Product Owner*

SUMÁRIO

1 INTRODUÇÃO

O cenário atual de desenvolvimento de *software* é marcado por uma constante busca por aprimoramento, eficiência e qualidade. Em um mundo cada vez mais digital e orientado por serviços, as organizações enfrentam o desafio de não apenas entregar produtos de *software* de alta qualidade, mas também garantir que esses produtos atendam às necessidades dinâmicas dos clientes e do mercado. Nesse contexto, a adoção de melhores práticas de gerenciamento e desenvolvimento de *software* se torna necessário [7].

O *Information Technology Infrastructure Library* (ITIL) [8] tem sido uma referência no que diz respeito ao gerenciamento de serviços de TI e práticas recomendadas [9]. Com o lançamento do ITIL4, ao enfatizar o *Service Value System* (SVS), um modelo que coloca a criação de valor no centro do gerenciamento de serviços de TI, houve uma significativa evolução nas abordagens e na compreensão do valor que a TI e os serviços de *software* podem proporcionar às organizações.

O ITIL4, adota algumas práticas para gerenciar serviços de TI. Uma prática de gestão é um conjunto de recursos organizacionais projetados para realizar um trabalho ou alcançar um objetivo. O ITIL4 inclui 34 práticas de gerenciamento. Para cada prática, há vários tipos de orientação, como termos e conceitos-chave, fatores de sucesso, atividades-chave ou objetos de informação [8].

No presente trabalho, focaremos na prática de Gerenciamento e Desenvolvimento de *Software*, cujo propósito é assegurar que os aplicativos atendam às necessidades das partes interessadas internas e externas. Os aplicativos de *software*, desenvolvidos internamente ou em parceria com um fornecedor, desempenham um papel crucial na entrega de valor aos clientes em negócios orientados por serviços tecnológicos. Por consequência, o desenvolvimento e gerenciamento de *software* é uma prática chave em qualquer organização de TI moderna, garantindo adequação dos aplicativos para suas finalidades e usos [8].

[10] Alega que o desenvolvimento de *software* representa uma atividade intensamente centrada no acúmulo de conhecimento, uma vez que a maioria dos projetos de desenvolvimento de *software* requer a colaboração de diferentes especialistas. Portanto, uma organização deve oferecer processos, métodos, técnicas e ferramentas como parte integrante do seu ambiente de produção. Vários autores [5] [11] destacam os benefícios significativos em termos de redução de tempo e custos de produção, bem como melhoria da qualidade, quando se adotam *frameworks* bem definidas.

De acordo com [8], os dois modelos de desenvolvimento aceitos para o desenvolvimento de *software* são referidas como modelo prescrito e ágil. O gerenciamento de *software*

é uma prática mais ampla, abrangendo as atividades contínuas de projetar, testar, operar e melhorar aplicativos de *software* para que continuem para facilitar a criação de valor.

No entanto, ainda não há consenso quanto a um processo de desenvolvimento de software que seja universalmente eficaz. Dito isto, algumas abordagens têm demonstrado sucesso em várias circunstâncias. Entre essas, destacam-se os processos, métodos e ferramentas alinhados com os valores e princípios delineados no Manifesto para o Desenvolvimento Ágil de Software [1] [12].

Este trabalho está dividido em uma fundamentação teórica, onde serão levantados conceitos e informações fundamentais para o entendimento e execução deste trabalho, abordando temas como governança de TIC e as principais estruturas de governança como o ITIL4 [8], detalhar o processo de desenvolvimento de software, elencando os principais modelos de processos de engenharia de *software*, como metologia cascata (*Waterfall*) e metodologias ágeis. Também faz parte da divisão do trabalho a metodologia, onde é demonstrado como o conjunto de informações adquiridos através da fundamentação teórica foi obtido e como ele vai afetar nos resultados obtidos que são apresentados no próximo capítulo, a conclusão. Nela será condensada as informações obtidas através dos estudos da fundamentação chegando ao objetivo final deste trabalho, que é apresentar a viabilidade e a necessidade da implementação de um *framework* para avaliar o grau de maturidade para práticas de gerenciamento e desenvolvimento de *software* baseado no ITIL4 aplicado às empresas de tecnologia, visando o sucesso da organização.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção serão descritos conceitos e informações fundamentais para o entendimento e execução deste trabalho. Primeiramente será apresentado a engenharia de *software* e o processo de gerenciamento e desenvolvimento de *software*, visto que estas são as áreas que norteiam todo o trabalho que estamos desenvolvendo do início ao fim. Na sequência, serão abordados os conceitos do ITIL 4, dando maior ênfase para a prática de gerenciamento e desenvolvimento de *software* visto que esta é também uma área foco do modelo. Logo após, abordaremos o processo de desenvolvimento de *software* GAIA e o seu alinhamento com o *framework* proposto. Posteriormente, serão apresentados modelos de referência no qual o estudo está se baseando/utilizando. Em seguida, será demonstrado os conceitos de níveis de maturidade ou capacidade, e a sua relação e importância com o modelo de maturidade desenvolvido no *framework* proposto.

2.1 Engenharia de *Software*

A atividade de desenvolvimento de *software* está em constante evolução, impulsionada por diversos fatores, tais como atualizações de tecnologia, mudanças nos ambientes operacionais, questões de segurança, demandas emergentes das organizações e dos usuários, entre outros. Este procedimento faz parte de um conjunto de processos integrados na ampla disciplina da engenharia de *software*.

Um processo envolve uma série de etapas ou atividades parcialmente organizadas com o propósito de alcançar um objetivo específico. Na engenharia de *software*, o objetivo primordial é assegurar a entrega previsível e eficiente de um produto de *software* capaz de satisfazer as necessidades e expectativas dos usuários ou da organização que solicitou o desenvolvimento desse *software* [13].

A engenharia de *software* é formada por um conjunto de vários processos, que quando combinados, executados, gerenciados, controlados e mantidos vão entregar um *software* de valor aos seus clientes/usuários.

De acordo com [14], a engenharia de software é crucial por duas razões fundamentais. A primeira decorre da crescente demanda de pessoas e organizações por sistemas de software avançados, o que impulsiona a engenharia de software a aprimorar sua capacidade de desenvolver soluções de forma cada vez mais confiável, eficiente e ágil. A segunda razão reside na constatação de que, a longo prazo, é mais econômico utilizar técnicas de engenharia de software em vez de simplesmente criar software de acordo com ideias imediatas. Essa economia se torna evidente quando se trata de realizar correções e manutenções necessárias nos sistemas de software.

Para [1], a engenharia de *software* se caracteriza como a aplicação de uma abordagem sistemática, quantificável e disciplinada no desenvolvimento, na operação e manutenção do *software* solicitado e que ainda é composta por várias camadas, incluindo o foco na qualidade, métodos, ferramentas e, segundo o autor, a camada fundamental é o processo. O processo é visto como o elemento unificador que mantém todas as camadas tecnológicas coesas e alinhadas. Além disso, é essa camada que viabiliza e promove o desenvolvimento de *software* de maneira lógica e dentro dos prazos estipulados.

O processo de *software* deve ser abordado e compreendido como algo flexível permitindo que a equipe do projeto selecione o conjunto mais adequado de ações e tarefas para realizar suas atividades com sucesso, não sendo uma prescrição rígida ou impositiva de como desenvolver um software.

O *framework* (metodologia) de processo genérico aplicado ao *software* abrange um conjunto de atividades de apoio essenciais, que podem ser empregadas em uma ampla variedade de projetos de *software*, independentemente de sua complexidade ou escala. Essa versatilidade é viável porque os processos de *software* podem variar de caso para caso, mas as atividades metodológicas essenciais permanecem constantes. Tais atividades estão descritas conforme [1]:

- **Comunicação:** Para dar início ao desenvolvimento de qualquer projeto, é crucial manter uma comunicação constante e colaborativa com os integrantes da equipe, a fim de compreender plenamente os objetivos das partes envolvidas e identificar as necessidades que irão contribuir para a definição das funcionalidades e características do *software*;

- **Planejamento:** A criação de *software* é uma atividade que requer meticulosidade em cada detalhe. A partir do momento em que elaboramos o planejamento, criamos uma espécie de guia que nos orienta em nossa jornada. Esse guia é conhecido como plano de projeto de *software*, que delineia o escopo da engenharia de *software* a ser realizada. Ele engloba as atividades técnicas a serem executadas, os recursos necessários, os possíveis riscos envolvidos, o cronograma e os produtos finais a serem entregues;

- **Modelagem:** Elaborar um rascunho do que será produzido, com a finalidade de obter uma visão abrangente do projeto como um todo. Essa etapa envolve a criação de modelos que visam a aprimorar a compreensão das necessidades dos clientes e resolver quaisquer pontos questionáveis, ausentes ou incorretos, entre outros aspectos;

- **Construção:** Elaboração do código (manualmente ou de forma automatizada) e execução dos testes para identificar eventuais erros que possam surgir, assegurando

que o que foi requisitado tenha sido devidamente implementado;

- **Emprego:** O *software* ou componente de *software* é entregue ao cliente que o utilizará e fornecerá um *feedback* de acordo com sua avaliação.

Aliado às atividades metodológicas, existem ainda as atividades de apoio, cuja finalidade, conforme o próprio termo indica, é dar suporte integral ao ciclo de desenvolvimento de *software*. Conforme [1], essas atividades são:

- **Controle e acompanhamento do projeto:** Permite que a equipe monitore o progresso do projeto em relação ao plano estabelecido e tome as ações necessárias para garantir a aderência ao cronograma;
- **Administração de riscos:** Avalia e aborda potenciais ameaças que possam afetar o desenvolvimento do projeto;
- **Garantia da qualidade de *software*:** Realiza a definição, supervisão, monitoramento e controle das atividades destinadas a assegurar a qualidade do *software*;
- **Revisões técnicas:** Realiza a avaliação dos artefatos de *software* produzidos, com o objetivo de identificar e corrigir potenciais erros e inconsistências antes que possam afetar as fases ou atividades subsequentes;
- **Medição:** Estabelece e recolhe métricas e indicadores do processo, do projeto, do produto ou do serviço, com a finalidade de facilitar a entrega do *software* com os requisitos adequados. Essa abordagem pode ser integrada às outras atividades metodológicas de suporte;
- **Gerenciamento da configuração de *software*:** Administra as alterações e suas consequências no âmbito do processo de desenvolvimento de *software*;
- **Gerenciamento da reusabilidade:** Estabelece critérios para a reutilização de artefatos, tais como documentos, requisitos e componentes de *software*, e implementa procedimentos para adquirir componentes reutilizáveis;
- **Preparo e produção de artefatos de software:** Abrange todas as tarefas essenciais relacionadas à criação de artefatos, como modelos, documentos, registros, formulários, listas, *checklists*, protótipos e outros recursos.

Além do que foi previamente apresentado, de acordo com [14], há quatro atividades essenciais que são compartilhadas por todos os processos de desenvolvimento de *software*:

- **Especificação de *software*:** Os clientes e engenheiros colaboram para estabelecer as especificações do *software* a ser desenvolvido, juntamente com seus respectivos impedimentos;
- **Desenvolvimento de *software*:** Quando o *software* é de fato projetado e programado;
- **Validação de *software*:** Quando o *software* é minuciosamente examinado e validado para assegurar que não contenha falhas críticas, ou que eventuais imperfeições estejam dentro dos limites aceitáveis em termos de qualidade do produto final, e também para confirmar que os requisitos especificados tenham sido adequadamente implementados;
- **Evolução de *software*:** Durante o ciclo de vida do software, são efetuadas alterações e procedimentos de manutenção.

Cada uma dessas etapas implica a ocorrência de erros, e a responsabilidade do engenheiro consiste em minimizá-los através da aplicação de princípios adequados à tarefa. De acordo com [15], os riscos mais frequentes associados a projetos de *software* incluem:

- Estourar cronograma;
- Estourar orçamento;
- Produto que não atende as expectativas do mercado e de baixa qualidade;
- Produtos não gerenciáveis, difíceis de manter e sem chance de evoluir.

O meta-papel do engenheiro de *software* é fornecer à equipe de desenvolvimento as ferramentas e processos necessários, garantindo a constante verificação da eficácia e otimização de seu uso [15]. Além disso, ele assegura a melhoria contínua dos processos que tornam a produção viável.

2.1.1 Controle de qualidade

A princípio, a noção de qualidade pode parecer intuitiva, mas ao investigar mais profundamente esse conceito, torna-se evidente que ele é muito mais abrangente. Quando tentamos definir a qualidade em relação a metas específicas a serem alcançadas, percebemos que esse conceito está longe de ser trivial, principalmente pelo fato de que o termo qualidade é subjetivo, ou seja, o que é qualidade para determinada pessoa, pode ser a falta da mesma para outro indivíduo. [16]

Conforme [1], é possível, em termos amplos e de forma geral, conceituar a qualidade de um *software* como a concretização dos requisitos funcionais e de desempenho

claramente especificados, bem como o cumprimento das normas de desenvolvimento devidamente documentadas e a incorporação das características subjacentes esperadas em todo *software* desenvolvido com padrões profissionais.

[17] define cinco ponto de vistas diferentes para definir o termo qualidade de *software*:

1. **Visão Transcedental:** aquela que se revela como algo difícil de descrever, mas que se torna reconhecível quando está presente;
2. **Visão do Usuário:** manifesta quando o software atende plenamente às necessidades do usuário;
3. **Visão do Desenvolvedor:** se concentra na conformidade com os padrões do processo como medida de qualidade;
4. **Visão do Produto:** direciona o foco para as características tangíveis do produto, considerando a qualidade interna como um fator crucial para garantir um desempenho de qualidade externa durante o uso.
5. **Visão baseada em Valor:** relacionada com o valor que os clientes atribuem a um produto de *software*, refletindo a sua aceitação e satisfação.

Para assegurar a qualidade e avaliar os procedimentos, [18] observa que, ao longo das últimas décadas, no âmbito da engenharia de *software*, houve uma concentração na melhoria dos processos de desenvolvimento de *software*, por meio da adoção de padrões como o CMMI, a integração de abordagens ágeis (principalmente XP e SCRUM) e a promoção da excelência da qualidade com base em *frameworks*, tais como o ISO 9126 e suas extensões, como o ISO 25000 [19] e o ISO 25060 [20].

2.2 Modelos de desenvolvimento prescritivos

Conforme [15], os modelos de processos prescritivo têm como finalidade descrever as atividades a serem realizadas no processo de desenvolvimento de *software*. Essa categoria de modelos visa transformar a atividade de criação de sistemas, que costumava ser um processo artesanal, em um procedimento bem estruturado, documentado e de alta qualidade, apto a ser incorporado às operações de produção empresarial. Esses processos são estruturados seguindo determinados padrões e abordagens, conhecidos como "ciclos de vida".

As metodologias tradicionais também são chamadas de "metodologias pesadas"[21], devido a sua característica de ênfase na documentação completa do software antes de sua implementação e pela sua relutância em se adaptar a mudanças durante o processo de

desenvolvimento. Em outras palavras, essas abordagens se comprometem a seguir o plano inicial do início ao fim [15]. Essa rigidez representa a principal limitação das metodologias tradicionais, uma vez que seu foco principal reside na previsibilidade dos requisitos do sistema. Assim, se houver qualquer alteração nas especificações, os processos de análise e design se tornam demorados e de difícil manutenção.

2.2.1 Modelo Cascata

O Modelo Cascata, também chamado de *Waterfall* sugere uma abordagem sequencial e sistemática para o desenvolvimento de *software*. [15] destaca que a origem deste modelo remete à década de 1970 e sua filosofia é fundamentada no conceito de BDUF (*Big Design Up Front*, "design completo antes de tudo", em português). Este princípio consiste em elaborar uma análise e design detalhados antes de iniciar a codificação. Desta forma, quando o código for efetivamente produzido, ele estará mais o próximo possível dos requisitos alinhados com o cliente. As etapas deste modelo estão ilustrados na figura 1, retirada de [1].

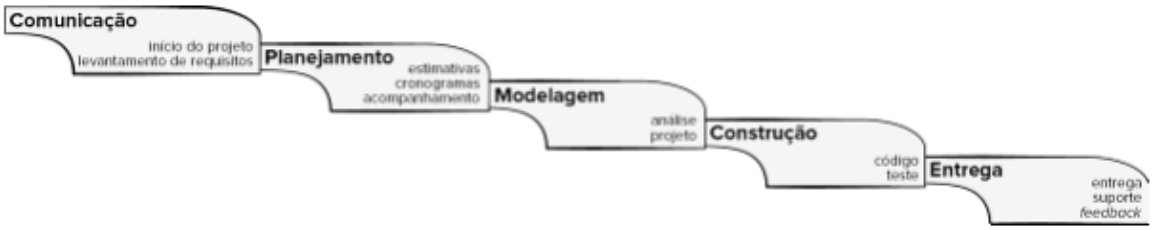

Figura 1 – Modelo Cascata

Ao final de cada fase, o *Waterfall* incorpora uma atividade de revisão, garantindo que o projeto progrida de maneira adequada para a próxima fase. Caso o projeto não esteja pronto para avançar, deverá permanecer na mesma fase. Embora o modelo tenha amplamente empregado ao longo de muitos anos, desde os primórdios da computação até recentemente, como aponta [22], devido ao aumento da complexidade dos sistemas e às demandas do mercado em constante evolução, ciclos de desenvolvimento mais adaptáveis e eficazes na gestão de erros, como os abordados nos métodos incremental e iterativo, estão se tornando mais predominantes.

[23] destaca que o modelo em cascata é vantajoso quando os requisitos do sistema são bem estabelecidos e/ou quando se trata de aprimorar um sistema já consolidado ou incorporar novas funcionalidades. No entanto, o autor aponta as principais desvantagens desse modelo no contexto atual da Engenharia de *Software* como:

- Projetos reais raramente seguem um fluxo linear, especialmente quando se trata

de lidar com exigências em constante mudança no mercado. Embora seja possível identificar iterações e adaptar-se a elas, o modelo como um todo não é facilmente flexível a mudanças. Portanto, a menos que cada etapa seja executada sem erros, o processo como um todo fica comprometido, e a fase de correções pode se tornar tão demorada quanto a etapa inicial de construção.

- O cliente não tem acesso imediato a um sistema operacional, uma vez que este só estará disponível após a conclusão de todas as iterações. Quando o cliente finalmente acessa o sistema, se os requisitos iniciais não estiverem alinhados com suas expectativas, os resultados podem ser desastrosos.

- O bloqueio no fluxo de produção entre membros da equipe é comum, pois eles muitas vezes precisam esperar que outros completem suas tarefas antes de poderem avançar com o trabalho. Isso leva a uma redução na produtividade e ao desperdício de recursos financeiros.

2.3 Modelos de desenvolvimento evolucionários

No processo de desenvolvimento de *software*, a criação de um projeto pode se estender por meses, e em alguns casos, até anos, antes de ser concluída. No entanto, frequentemente, o fim do desenvolvimento não marca o término do trabalho, mas sim o início de uma nova fase dedicada à manutenção e à adaptação contínua às mudanças no mercado tecnológico.

Os modelos evolucionários são os responsáveis por começarem a abordar a ideia de como lidar com mudanças, uma vez que os requisitos de negócios e de produtos podem sofrer alterações frequentes durante o curso do desenvolvimento da aplicação. Tem-se então a concepção da possibilidade de desenvolver produtos que possam evoluir ao longo do tempo.

2.3.1 Modelo incremental

De acordo com [22], o processo incremental está associado à noção de ampliar gradualmente a área de um sistema. A meta é colaborar com o usuário de maneira progressiva até que se obtenha o produto final desejado. O modelo incremental se revela especialmente relevante quando não é viável definir completamente o escopo dos requisitos iniciais e mostra resultados positivos ao ser aplicado em sistemas de pequena escala, onde a quantidade de informações e, portanto, de erros e riscos, é significativamente reduzida.

O modelo incremental surgiu como uma alternativa para suprir alguns dos problemas do Modelo Cascata, combinando elementos de seu predecessor com etapas interativas, cujo objetivo é apresentar um produto operacional a cada incremento realizado.

Nesse sentido, [22] sugere que seja uma prática intuitiva dividir o trabalho em partes mensuráveis, chamadas de iterações. Ao final de cada iteração, o produto em desenvolvimento recebe um incremento, ou seja, uma adição ou aprimoramento, contribuindo assim para o desenvolvimento incremental do projeto. Dessa forma, a equipe concentra-se na melhoria constante dos resultados e dos processos.

As principais vantagens do modelo, de acordo com [1] são:

- Possibilitar uma implementação e entrega rápida de um *software* útil ao cliente final, mesmo que todas as funcionalidades não sejam incluídas inicialmente;
- Os clientes têm a oportunidade de utilizar e beneficiar-se do *software* primordial mais cedo do que seria possível através de um processo em cascata;
- É mais simples coletar *feedback* dos clientes em relação ao progresso alcançado no desenvolvimento. Os clientes têm a oportunidade de avaliar as demonstrações do *software* e observar o grau de implementação alcançado;
- A quantidade de análise e documentação a ser refeita é muito menor do que o necessário no modelo em cascata.

Apesar das vantagens apontadas, [14] aponta dois problemas principais no modelo incremental:

- O progresso não é visível, e os gestores necessitam de entregas periódicas para avaliar o andamento. Se os sistemas forem desenvolvidos com rapidez, não será economicamente justificável criar documentos que descrevam cada uma das versões do sistema;
- A adição de novos incrementos tende a provocar a degradação da estrutura do sistema, o que, por sua vez, demanda alocação de tempo e recursos financeiros para realizar refatorações e aprimoramentos no *software*. Isso ocorre porque as mudanças constantes, sem previsibilidade do futuro, tendem a comprometer a integridade da sua estrutura. Consequentemente, a incorporação de alterações no *software* torna-se progressivamente mais custosa e complexa.

2.3.2 Modelo de prototipação

Em relação ao modelo de prototipagem, ele emprega o conceito de prototipação, com o proposito de desenvolver, de forma ágil e econômica, versões de interface do projeto para que sejam avaliadas e testadas diversas alternativas pelo cliente antes da entrega do produto.

Segundo [1], o paradigma da prototipação é particularmente vantajosa em cenários nos quais o cliente não consegue definir detalhadamente os requisitos das funcionalidades e recursos desejados, ou ainda, quando o programador está inseguro quanto à eficácia do algoritmo implementado ou quando há dúvidas quanto à adaptabilidade do sistema operacional e a interação homem-máquina.

Embora o modelo de prototipação possa ser empregada como um modelo de processo independente, é frequentemente utilizada como uma alternativa viável que pode ser incorporada em qualquer metodologia. Em termos gerais, os protótipos podem ser desenvolvidos de maneira "descartável"ou seguindo uma abordagem evolutiva, na qual eles passam por um processo de evolução gradual até se transformarem no produto final.

2.3.3 Modelo espiral

O Modelo espiral concentra as qualidade mais vantajosas do ciclo de vida clássico e de prototipação, agregando um componente adicional: a análise riscos [24]. Ele usa uma abordagem “evolucionária” que capacita tanto os desenvolvedores quanto os clientes a compreenderem e reagirem aos riscos em cada fase evolutiva, Aproveitando-se da prototipagem como uma ferramenta para mitigar riscos, ele também oferece a flexibilidade de empregar essa abordagem em qualquer estágio do desenvolvimento do projeto.

A figura 2, retirada de [1], ilustra algumas etapas do modelo.

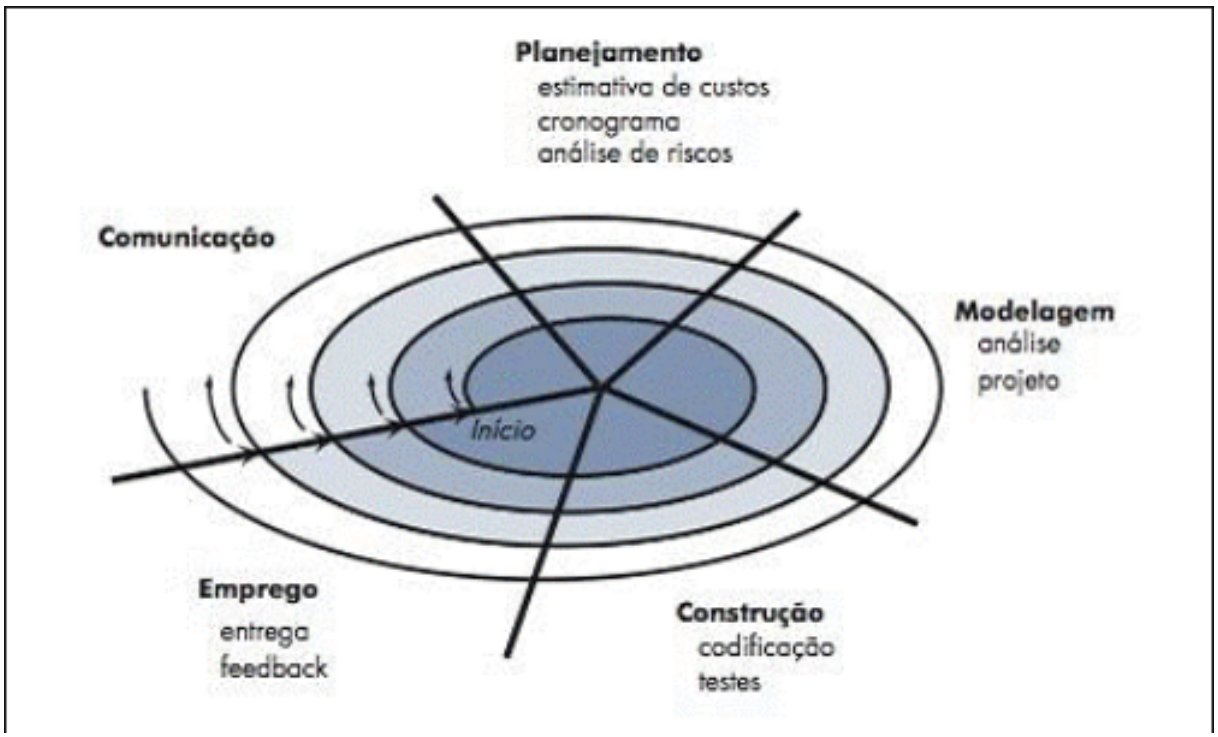

Figura 2 – Modelo Espiral (Fonte: [1])

Sempre começamos no centro da espiral e continuamos no sentido horário. Os riscos são avaliados à medida que avançamos em cada iteração. A primeira etapa envolve a criação de uma especificação de produto, as etapas subsequentes podem ser usadas para desenvolver um protótipo e, à medida que avançamos, evoluímos para versões cada vez mais avançadas do *software*. Cada passo no planejamento, por exemplo, resulta em ajustes no plano do projeto. O custo e o cronograma são sempre adaptados com base no *feedback* do cliente após cada entrega. Além disso, o número de iterações planejadas para concluir o *software* também será ajustado.

[15] observa que, ao contrário de outros modelos nos quais o encerramento ocorre após a entrega do software, o modelo em espiral pode ser ajustado a cada entrega. O projeto é concluído quando o cliente se encontra plenamente satisfeito, quando o software é retirado de operação ou quando uma data específica determina o encerramento definitivo do projeto.

Apesar de vantajoso, [15] ainda aponta para o fato deste modelo ter um alto nível de complexidade, necessitando ser gerenciado eficientemente em total controle do processo e que apenas projetos de grande porte, com alto nível de risco, requisitos pouco conhecidos e um alto grau de originalidade, como o desenvolvimento de jogos eletrônicos, podem colher os benefícios desse modelo.

2.4 Modelos de desenvolvimento ágil

Em 2001, um grupo formado por dezessete pesquisadores e profissionais de TI se reuniram para discutir as técnicas existentes para projetos de desenvolvimento de *software*. O objetivo deles era descobrir novos valores e princípios para melhorar o desenvolvimento de projetos. O acontecimento mais importante em relação à este movimento foi a assinatura do Manifesto Ágil. [12]

Conforme [14] aponta, as metodologias ágeis se destacam por sua abordagem iterativa na criação de *software*, na qual os requisitos, o processo de desenvolvimento e os testes são entrelaçados. Isso significa que o *software* é disponibilizado de maneira incremental ao longo do processo, focando nos pontos críticos.

Os valores e princípios ágeis promovidos por essa aliança e a sua relação com os valores tradicionais foram apresentados com o proposito de auxiliar as equipes e aprimorar entrega de produtos de *software*. Com a popularização desses métodos, passaram a ser utilizados em diversas outras áreas de negócio, como criação de empresas e desenvolvimento de produtos. Os valores descritos no Manifesto e considerados os pilares para os métodos ágeis são: [12]

1. **Indivíduos e interações** mais que processos e ferramentas;

2. ***Software* em funcionamento** mais que documentações abrangentes;
3. **Colaboração com o cliente** mais que negociação de contratos;
4. **Responder a mudanças** mais que seguir um plano.

Além da adoção do processo iterativo e dos 4 valores, o desenvolvimento ágil também se baseia em 12 princípios estabelecidos em [12], sendo estes:

1. **Satisfação do Cliente:** Priorização da satisfação do cliente entregando *software* com valor agregado de forma contínua e adiantada;
2. **Adaptação às Mudanças:** Aceite de mudanças nos requisitos, mesmo em estágios avançados de desenvolvimento, buscando vantagens competitivas para o cliente.
3. **Entrega Frequentemente:** Entregas frequentes de *software* funcional em ciclos curtos, preferencialmente em poucas semanas;
4. **Colaboração Diária:** Estabelecimento de colaboração diária entre pessoas de negócios e desenvolvedores ao longo do projeto.
5. **Motivação e Confiança:** Projetos construídos em torno de equipes motivadas, fornecendo o ambiente e o suporte necessários e confiando nelas para fazer o trabalho.
6. **Comunicação Face a Face:** A comunicação mais eficaz ocorre através de conversas presenciais.
7. **Software Funcional:** O software funcional é a principal medida de progresso.
8. **Desenvolvimento Sustentável:** Promoção do desenvolvimento sustentável, permitindo que patrocinadores, desenvolvedores e usuários mantenham um ritmo constante.
9. **Excelência Técnica:** Manter atenção constante à excelência técnica e ao bom design para aumentar a agilidade.
10. **Simplicidade:** Valorização da simplicidade, maximizando a quantidade de trabalho não realizado.
11. **Auto-organização:** As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. **Reflexão e Ajuste:** Em intervalos regulares, a equipe reflete sobre sua eficácia e faz ajustes em seu comportamento para melhorar continuamente.

Na Tabela 1, adaptada de um estudo de caso por [5], pode-se observar os benefícios de adoção de uma metodologia em relação a outra e seu impacto no processo produtivo.

Tabela 1 – Metologias tradicionais x ágil. (Fonte: [5])

Critério de Qualidade	**Método Tradicional**	**Desenvolvimento Ágil**
Tempo de Desenvolvimento	Desenvolvedores trabalham na produção de todos os requisitos	Desenvolvedores trabalham em um único requisito
***Bugs* encontrados**	A medida que uma considerável parte do software é concluída, é comum identificarmos uma grande quantidade de erros e defeitos durante a realização dos testes.	Após a conclusão de cada recurso, são realizados testes as devidas correções, resultando em uma redução significativa na incidência de erros.
Tempo para entrega	Após término de todos os requisitos	Após finalizar um requisito
Aplicação teste	Testes são realizados somente no término	Pequenas partes são submetidas a testes, e os resultados são prontamente encaminhados aos desenvolvedores
Documentação	O resultado é adequadamente documentado. As especificações são documentadas antes do início da produção. O trabalho de produção é documentado após sua conclusão.	Os requisitos são documentados. Ferramentas automatizadas produzem informações e estatísticas acerca dos recursos desenvolvidos, dos resultados dos testes e dos problemas identificados.
Gerenciamento de Mudanças	Alterações passam por um extenso processo de revisão e aprovação antes de serem aceitas. As mudanças são incorporadas por meio de um *patch*, que é um arquivo compactado contendo um conjunto abrangente de modificações.	Durante o processo, novos requisitos são acolhidos e incorporados como mudanças a pedido do cliente.
Modelo de custo	A construção e integração de um software demandam tempo e esforço consideráveis quando realizados manualmente.	Custo associado a ferramentas e servidores para a prática de desenvolvimento ágil

É importante destacar que as Metodologias Ágeis não descartam completamente os processos, ferramentas, documentação e negociações. No entanto, elas lhes atribuem

uma importância secundária. O método ágil busca constantemente o MVP (*Minimum viable product*, ou Produto mínimo viável em português) como ponto de partida, ciente de que podem e irão aprimorá-lo posteriormente.

2.4.1 Scrum

Scrum, segundo [25] é um *framework* fundamentado nos valores e princípios do Manifesto Ágil, possui grande destaque no mercado, principalmente voltado para o desenvolvimento de *software*, devido à sua relativa facilidade de implementação e à sua capacidade de superar os obstáculos enfrentados pelas equipes de desenvolvimento.

Seu conceito fundamenta-se na premissa de que o processo de desenvolvimento de *software* é altamente imprevisível, devido às inúmeras variáveis técnicas e ambientais que podem ser modificadas ao longo do ciclo de desenvolvimento. Isso resulta em uma considerável quantidade de incertezas associadas a esses projetos. Portanto, o Scrum concentra seus esforços em encontrar uma abordagem flexível para criar um produto de *software*, reconhecendo que as mudanças e desafios inerentes ao produto final são uma constante [26].

O Scrum parte do conceito de empirismo, que é a concepção de que o conhecimento é adquirido por meio da experiência e da tomada de decisões fundamentadas no que já se sabe. De acordo com [27], existem três pilares que apoiam a implementação de controle do processo empírico: transparência, inspeção e adaptação. Transparência refere-se à visibilidade dos elementos e dos resultados que devem estar acessíveis aos responsáveis pelo processo. A inspeção identifica e orienta a gestão das variações nos processos e nos artefatos. A adaptação engloba os ajustes requeridos nos processos e nos artefatos do *framework*.

Figura 3 – Processo Scrum (Fonte: [2])

No contexto do desenvolvimento de *software* com base no Scrum, representado na figura 3 é possível identificar três níveis cruciais: *Sprint Planning*, *Release Planning* e *Product*. Estes níveis desempenham um papel fundamental na formulação de uma lista de funcionalidades, organizadas por ordem de prioridade. Essa etapa é crucial para a definição do que se tornará o produto, comumente também chamado de *Product Backlog* [28].

O *Product Backlog* contém, em ordem de prioridade, as funcionalidades do produto ou do negócio. Esse *backlog* é dividido em subconjuntos conhecidos como *Releases*, que representam entregas parceladas do produto completo. Os *Releases* podem ser subdivididos ainda mais em *Sprints*, que de forma sucinta, considera-se como sendo um pequeno projeto integrado ao projeto principal, com uma duração previamente estabelecida. Dentro desse prazo determinado, a expectativa é alcançar ou se aproximar ao máximo da conclusão de todas as tarefas priorizadas pelo *Product Owner* (PO), que representa o cliente no time [28].

O time, por sua vez, é constituído basicamente em três papéis distintos, sendo eles: o *Scrum Master* (ou líder do projeto), que possui a função de facilitador, frequentemente atuando como mediador nas interações da equipe; o *Product Owner* (como já visto, representa o cliente) e o *Dev Team* (equipe de desenvolvimento).

Os objetivos de cada *Sprint* e o *Scrum Team* não devem sofrer alterações ao longo da *Sprint* em curso. No entanto, tanto o PO quanto a equipe Scrum têm a flexibilidade de ajustar o escopo do projeto, conforme necessário. Além disso, o PO tem a prerrogativa de cancelar uma *Sprint*, caso ocorram mudanças significativas na direção da empresa, nas demandas do mercado ou nas necessidades tecnológicas. O trabalho realizado durante uma *Sprint* é ajustado de acordo com o problema em questão, permitindo que seja facilmente modificado pela equipe, sem qualquer obstáculo. A fim de possibilitar esse cenário, realizam-se reuniões breves diariamente, nas quais os membros da equipe têm a oportunidade de expor problemas, sugerir ideias e compartilhar qualquer informação relevante para o andamento do processo [29].

É importante destacar que no contexto do desenvolvimento ágil com Scrum, cada integrante da equipe desempenha papéis dinâmicos. Isso não implica a ausência de responsabilidades individuais, mas sim a liberdade de colaborar, compartilhar informações, auxiliar uns aos outros e solucionar desafios coletivos. No término de cada *Sprint*, ocorrem encontros mais aprofundados, nos quais se expõe o trabalho realizado ao longo da *Sprint*, promove-se a discussão de *feedbacks* e, com base nas lições aprendidas, elabora-se o planejamento para a próxima *Sprint*.

2.4.2 *Extreme Programming*

Segundo [30], o XP (*Extreme Programming*) é uma metodologia ágil, geralmente praticada em times pequenos (geralmente de 12 a 14 pessoas) que ajuda a produzir *softwares* de alta qualidade e facilitar a vida do time de desenvolvimento. Esta metodologia recebe esse nome, pois as práticas tradicionais de programação são levadas a níveis extremos. O propósito básico de desenvolver este modelo era criar um modelo de processo leve, ajudando a criar um *software* de acordo com os requisitos do usuário. Da mesma forma que o Scrum, o XP também se baseia inteiramente no manifesto ágil e compartilha etapas e características muito similares. Uma distinção notável entre eles pode ser observada na maior área de atuação do Scrum, que abrange diversas áreas, enquanto o XP está focado exclusivamente no desenvolvimento de *software*.

De acordo com [31], a metodologia XP é baseada em cinco valores principais, sendo eles:

1. **Simplicidade:** O XP recomenda que cada integrante da equipe opte pela solução mais simples que seja eficaz. O propósito é realizar o que é mais simples no presente e estabelecer um ambiente em que os custos de futuras mudanças sejam reduzidos. O intuito dessa abordagem é evitar a construção prematura de funcionalidades, diferentemente do que ocorre em diversas metodologias tradicionais, as quais frequentemente resultam em funcionalidades que acabam por não serem utilizadas;

2. **Comunicação:** Concentrar-se em desenvolver uma compreensão pessoal do problema, minimizando o uso de documentação formal e maximizando a interação direta entre as pessoas que participam do projeto. As práticas da XP, como programação em pares, testes e comunicação com o cliente, visam promover a comunicação eficaz entre gerentes, programadores e clientes.

3. ***Feedback*:** Os programadores recebem *feedback* sobre a lógica de seus programas ao escreverem e executarem casos de teste. Por sua vez, os clientes obtêm *feedback* por meio dos testes funcionais que são desenvolvidos para todas as histórias (casos de uso simplificados). Essa retroalimentação desempenha um papel crucial, uma vez que permite que as pessoas aprimorem gradualmente seu entendimento do sistema, identifiquem e corrijam erros, e façam melhorias no sistema;

4. **Respeito:** Todos dão e sentem o respeito que merecem como membros valiosos da equipe. Todos contribuem com valor, mesmo que seja simplesmente entusiasmo. Os desenvolvedores respeitam a expertise dos clientes e vice-versa. A administração respeita o direito de aceitar responsabilidade e receber autoridade sobre o trabalho desempenhado;

5. **Coragem:** Ela é necessária para que realmente se aplique XP como deve ser aplicado. Alguns exemplos de atitudes que demandam coragem incluem: modificar código que já está em funcionamento, descartar todo o código e reescrevê-lo do zero, e permitir o compartilhamento de código por toda a equipe. Essas ações podem ser essenciais para aprimorar o projeto e não devem ser evitadas apenas por receio de enfrentá-las.

Além dos valores, o XP ainda se baseia em treze práticas importantes, listadas na tabela 2, adaptada de [6]:

Tabela 2 – Práticas que compõem o XP. (Fonte: [6])

Prática no XP	**Descrição**
Cliente presente	O método XP parte da premissa de que o cliente deve liderar o processo de desenvolvimento com base no feedback recebido do sistema.
Jogo do planejamento	No começo de cada ciclo de iteração, temos o momento do planejamento, que consiste em uma reunião na qual o cliente avalia as funcionalidades a serem desenvolvidas.
Stand Up Meeting	Todas as manhãs, a equipe se reúne para avaliar o progresso feito no dia anterior e determinar as prioridades para o dia que se inicia.
Programação em pares	Os desenvolvedores colaboram em duplas, o que significa que em frente a cada computador, há sempre duas pessoas trabalhando juntas para criar o código correspondente.
Desenvolvimento guiado para testes	Antes de codificar cada funcionalidade, é prática comum escrever testes específicos para assegurar um entendimento mais aprofundado das necessidades do cliente.
Refatoração	Refatorar é o processo de modificar o código sem impactar a funcionalidade que ele oferece, visando simplificar a manutenção do *software*.
Código coletivo	Os desenvolvedores possuem pleno acesso a todas as seções do código e têm a capacidade de realizar alterações que considerem relevantes, sem a necessidade de solicitar autorização de terceiros.
Código padronizado	Com o objetivo de simplificar a manutenção do código para toda a equipe, é estabelecido padrões de codificação que tornam o sistema mais uniforme e capacitam qualquer membro da equipe a realizar a manutenção do sistema.
Design simples	A fim de possibilitar ao cliente receber *feedback* rapidamente, é essencial que a equipe seja ágil no processo de desenvolvimento, o que, por sua vez, resulta na escolha pela simplicidade no *design*.
Metáfora	Com o objetivo criar um *design* simples, o time de desenvolvimento emprega metáforas, visto que estas possuem a capacidade de comunicar conceitos complexos de maneira simplificada.
Ritmo sustentável	Para assegurar que a equipe alcance consistentemente o seu desempenho máximo e desenvolva *software* da mais alta qualidade possível, a metodologia XP preconiza que os desenvolvedores limitem suas jornadas de trabalho a apenas oito horas por dia, evitando horas extras. Isso se deve à importância de estarem descansados a cada manhã, garantindo o pleno uso de suas capacidades mentais.
Integração contínua	Prática empregada para verificar ou testar uma aplicação toda vez que uma nova funcionalidade é adicionada, podendo ser realizado manualmente ou automaticamente, fazendo uso de ferramentas especializadas para esse fim.
Releases curtos	O principal propósito do XP é criar um fluxo ininterrupto de valor para o cliente. Nesse sentido, a abordagem envolve a criação de lançamentos curtos, nos quais a equipe desenvolve um conjunto limitado de funcionalidades e as implementa de forma ágil.

2.5 ITIL 4

A ITIL (*Information Technology Infrastructure Library*) foi criada pelo governo britânico no *Office of Government Commerce* (OGC), na década de 1980 com o objetivo de estabelecer um padrão para o gerenciamento dos processos da área de Tecnologia da Informação (TI) de seus departamentos. A princípio esse método seria utilizado pelas organizações do setor público a fim de garantir bons resultados tanto na qualidade quanto no custo. Segundo [32], a ITIL preocupa-se, basicamente, com a entrega e o suporte aos serviços de forma apropriada e aderente aos requisitos do negócio, é o modelo de referência para gerenciamento dos serviços de TI mais aceito mundialmente.

[33] diz que a ITIL não define os processos para serem implementados na área de TI na empresa, mas oferece uma base para colocar os processos já existentes em um contexto estruturado, validando tarefas, atividades, procedimentos e regras da organização. Tais práticas podem ser adotadas da forma que melhor atender às necessidades de cada organização.

A quarta edição do ITIL apresenta um modelo operacional que facilita a entrega de produtos e serviços tecnológicos. Esta edição incorpora abordagens contemporâneas e flexíveis em comparação com sua predecessora, a ITILv3 (2011), alinhando-se com as tendências atuais do mercado. Ela também integra práticas modernas de gestão de serviços, como *Agile*, *DevOps* e *Lean*, para uma abordagem mais dinâmica e eficaz [8].

2.5.1 Sistema de Valor de Serviço

Os elementos centrais da ITIL4 compreendem o Sistema de Valor de Serviço (SVS) e o modelo de Quatro Dimensões. O SVS detalha como os diversos componentes e atividades de uma organização operam de forma sinérgica para promover a geração de valor em relação aos serviços que envolvem Tecnologia da Informação. Os componentes principais do SVS são: [8]

- a cadeia de valor de serviços;
- as práticas ITIL;
- os princípios orientadores ITIL;
- governança;
- melhoria contínua.

A figura 4 retirada de [3] a seguir, ilustra o SVS

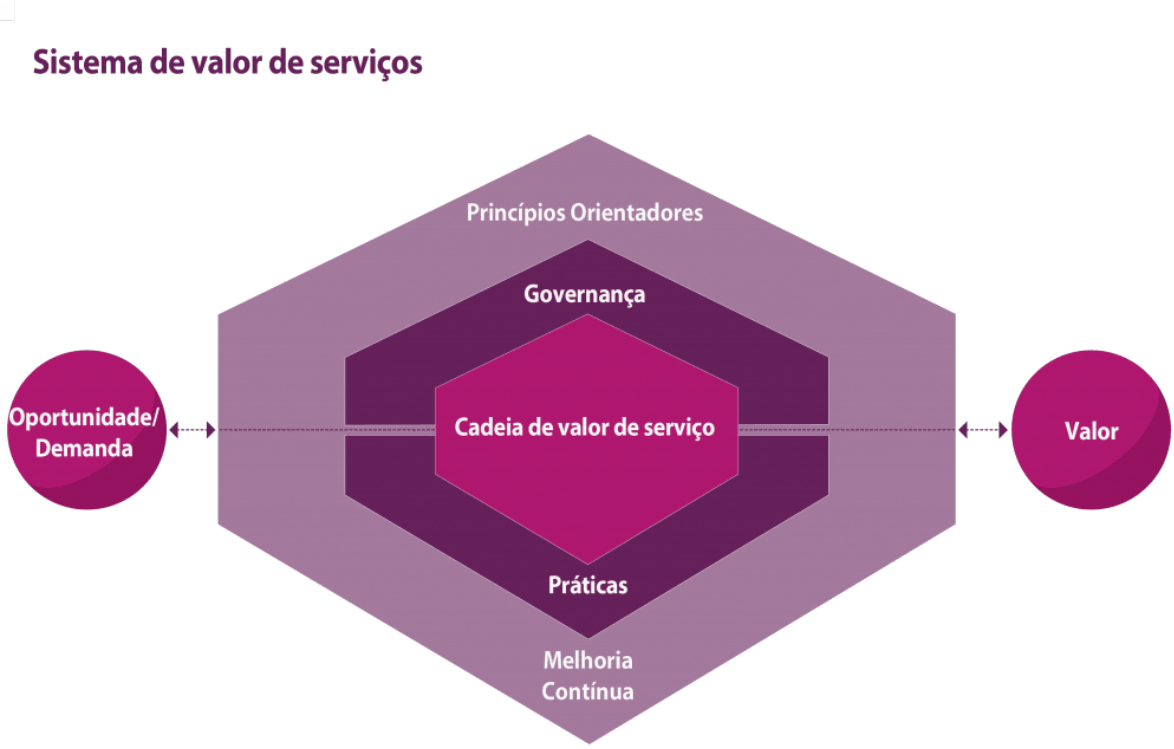

Figura 4 – SVS ITIL4. (Fonte [3])

O núcleo da SVS é a cadeia de valor de serviços, um modelo operacional flexível destinado à concepção, prestação e aperfeiçoamento contínuo de serviços. A cadeia de valor dos serviços engloba seis atividades fundamentais, conforme ilustrado na figura 5.

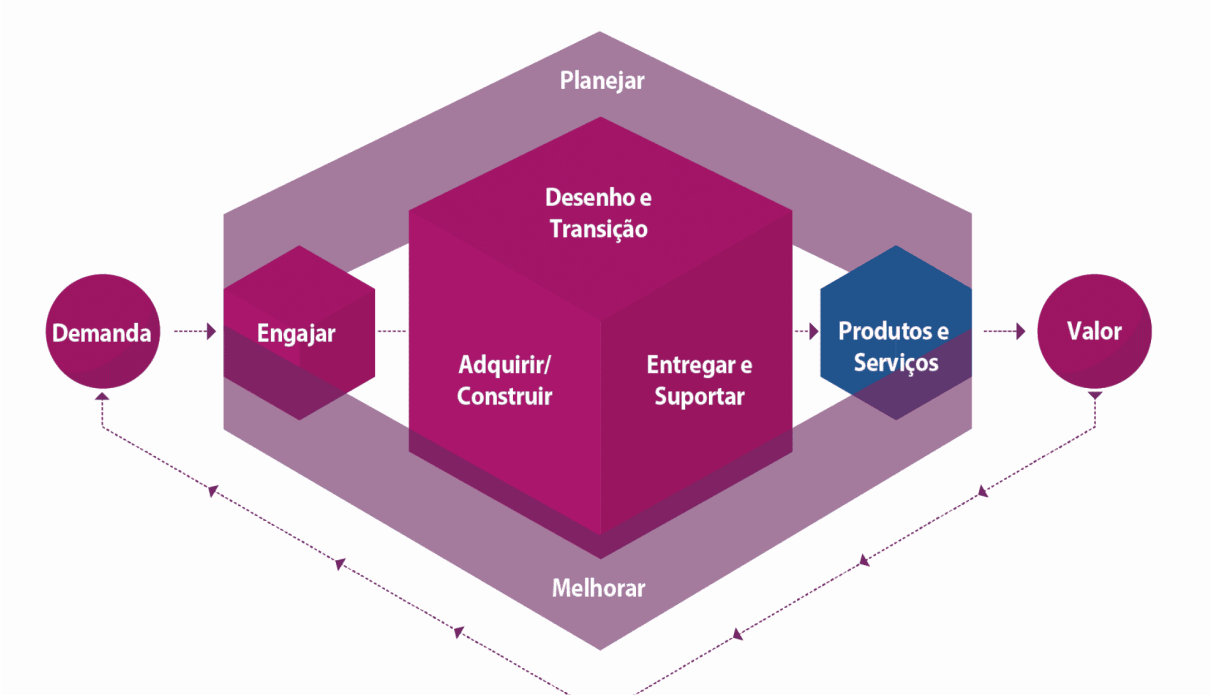

Figura 5 – Cadeia de Valor de Serviço ITIL4. (Fonte [3])

As atividades da Cadeia de Valor de Serviço podem ser combinados de diversas maneiras diferentes, possibilitando que uma organização crie várias versões de fluxos de valor. A adaptabilidade da cadeia de valor de serviço capacita uma organização a responder com eficácia e eficiência às mudanças nas necessidades das partes interessadas.

2.5.1.1 Planejar

Tabela 4 – Atividade Planejar da Cadeia de Valor de Serviço.

Entrada	**Saída**
• Políticas, requisitos e restrições fornecidos pelo órgão de governança da organização; • Demandas consolidadas e oportunidades fornecidas por **engajar**; • Informações de desempenho da cadeia de valor, relatórios de status de melhoria e iniciativas de **melhoria**; • Conhecimento e informações sobre produtos e serviços novos e alterados de ***design*** **e transição** e **adquirir/construir**; • Conhecimento e informações sobre componentes de serviços de terceiros de **engajar**.	• Planos estratégicos, táticos e operacionais; • Decisões de portfólio para ***design*** **e transição**; • Arquiteturas e políticas para ***design*** **e transição**; • Oportunidades de melhoria para **melhorar**; • Um portfólio de produtos e serviços para **engajar**; • Requisitos de contrato e acordo para **engajar**.

2.5.1.2 Melhorar

Tabela 6 – Atividade Melhorar da Cadeia de Valor de Serviço.

Entrada	**Saída**
• Informações de desempenho de produtos e serviços fornecidas por **adquirir/construir**; • *Feedback* de partes interessadas fornecido por **engajar**; • Informações de desempenho e oportunidades de melhoria fornecidas por **todas as atividades da cadeia de valor**; • Conhecimento e informações sobre produtos e serviços novos e alterados de ***design*** **e transição** e **adquirir/construir**; • Conhecimento e informações sobre componentes de serviços de terceiros de **engajar**.	• Iniciativas de melhoria para **todas as atividades da cadeia de valor**; • Informações sobre o desempenho da cadeia de valor para **planejar** e o corpo organizacional; • Relatórios de status de melhoria para **todas as atividades da cadeia de valor**; • Requisitos de contrato e acordo para **engajar**; • Informações sobre o desempenho do serviço para ***design*** **e transição**.

2.5.1.3 Engajar

Tabela 8 – Atividade Engajar da Cadeia de Valor de Serviço.

Entrada	**Saída**
• Um portfólio de produtos e serviços fornecido por **planejar**; • Requisitos detalhados para serviços e produtos fornecidos pelos clientes internos e externos; • Incidentes, solicitações de serviço e *feedback* dos usuários e clientes; • Informações sobre a conclusão de tarefas de suporte ao usuário provenientes da **entrega e suporte**; • Oportunidades de marketing de clientes e usuários atuais e potenciais; • Oportunidades de cooperação e *feedback* fornecidos por parceiros e fornecedores; • Requisitos de contrato e acordo de **todas as atividades da cadeia de valor**; • Conhecimento e informações sobre produtos e serviços novos e modificados provenientes do ***design* e transição**, e **adquirir/construir**; • Conhecimento e informações sobre componentes de serviço de terceiros de fornecedores e parceiros; • Informações sobre o desempenho de produtos e serviços provenientes de **entrega e suporte**; • Iniciativas e relatórios de status de melhoria provenientes da **melhorar**;	• Demandas consolidadas e oportunidades para **planejar**; • Requisitos de produtos e serviços para ***design* e transição**; • Tarefas de suporte ao usuário para **entrega e suporte**; • Oportunidades de melhoria e *feedback* dos *stakeholders* para **melhorar**; • Solicitações de início de projeto ou mudança para **adquirir/construir**; • Contratos e acordos com fornecedores externos e parceiros internos para ***design* e transição** e **adquirir/construir**; • Conhecimento e informações sobre componentes de serviços de terceiros para **todas as atividades da cadeia de valor**; • Relatórios de desempenho de serviços para clientes.

2.5.1.4 *Design* e transição

Tabela 10 – Atividade *Design* e transição da Cadeia de Valor de Serviço.

Entrada	**Saída**
• Decisões de portfólio fornecidas por **planejar**; • Arquiteturas e políticas fornecidas por **planejar**; • Requisitos de produtos e serviços fornecidos por **engajar**; • Iniciativas de melhoria fornecidas por **melhorar**; • Relatórios de status de melhoria de **melhorar**; • Informações de desempenho de serviços fornecidas por **entrega e suporte** e **melhorar**; • Componentes de serviço de **adquirir/construir**; • Conhecimento e informações sobre componentes de serviço de terceiros de **engajar**; • Conhecimento e informações sobre produtos e serviços novos e alterados de **adquirir/construir**; • Contratos e acordos com fornecedores externos e parceiros internos fornecidos por **engajar**.	• Requisitos e especificações para **adquirir/construir**; • Requisitos de contrato e acordo para **engajar**; • Novos produtos e serviços e produtos alterados para **entrega e suporte**; • Conhecimento e informações sobre produtos e serviços novos e alterados para **todas as atividades da cadeia de valor**; • Informações de desempenho e oportunidades de melhoria para **melhorar**.

2.5.1.5 Adquirir/Construir

Tabela 12 – Atividade Adquirir/Construir da Cadeia de Valor de Serviço.

Entrada	**Saída**
• Arquiteturas e políticas fornecidas por **planejar**; • Contratos e acordos com fornecedores externos e internos e parceiros fornecidos por **engajar**; • Bens e serviços fornecidos por fornecedores externos e internos e parceiros; • Requisitos e especificações fornecidos pelo ***design* e transição**; • Iniciativas de melhoria fornecidas por **melhorar**; • Relatórios de status de melhoria de **melhorar**; • Solicitações de iniciação de mudança ou projeto fornecidas por **engajar**; • Solicitações de mudança fornecidas pela **entrega e suporte**; • Conhecimento e informações sobre produtos e serviços novos e alterados de ***design* e transição**; • Conhecimento e informações sobre componentes de serviços de terceiros da participação.	• Componentes de serviço para **entrega e suporte**; • Componentes de serviço para ***design* e transição**; • Conhecimento e informações sobre componentes de serviço novos e alterados para **todas as atividades da cadeia de valor**; • Requisitos de contrato e acordo para **engajar**; • Informações de desempenho e oportunidades de melhoria para **melhorar**.

2.5.1.6 Entrega e Suporte

Tabela 14 – Atividade Entrega e Suporte da Cadeia de Valor de Serviço.

Entrada	**Saída**
• Novos produtos e serviços criados por ***design* e transição**; • Componentes de serviço fornecidos por **adquirir/construir**; • Iniciativas de melhoria fornecidas por **melhorar**; • Relatórios de status de melhoria de **melhorar**; • Tarefas de suporte ao usuário fornecidas por **engajar**; • Conhecimento e informações sobre novos componentes de serviço e serviços alterados de design e transição e **adquirir/construir**; • Conhecimento e informações sobre componentes de serviço de terceiros de **engajar**.	• Serviços entregues aos clientes e usuários; • Informações sobre a conclusão das tarefas de suporte ao usuário para **engajar**; • Informações sobre o desempenho de produtos e serviços para **engajar** e **melhorar**; • Oportunidades de melhoria para **melhorar**; • Requisitos de contrato e acordo para o **engajar**; • Solicitações de alteração para **adquirir/construir**; • Informações sobre o desempenho de serviços para ***design* e transição**.

2.5.2 Quatro dimensões do gerenciamento de serviços

O ITIL4 ainda define um sistema de gerenciamento de serviços de quatro dimensões, sendo elas críticas para a geração de valor para os clientes, enumeradas a seguir: [8]

- **Organizações e pessoas:** uma organização necessita de uma cultura que respalde seus objetivos, bem como do nível adequado de habilidades e competência dentro de sua equipe de colaboradores;
- **Informação e tecnologia:** dentro do contexto da SVS, isso abrange informações e o conhecimento, além das tecnologias essenciais para o gerenciamento de serviços.
- **Parceiros e fornecedores:** refere-se aos relacionamentos de uma organização com as outras empresas envolvidas no design, na implantação, na entrega, no suporte e na melhoria contínua dos serviços;

- **Fluxos de valores e processos:** como as várias partes da organização trabalham de forma integrada e coordenada é importante para permitir a criação de valor através de produtos e serviços.

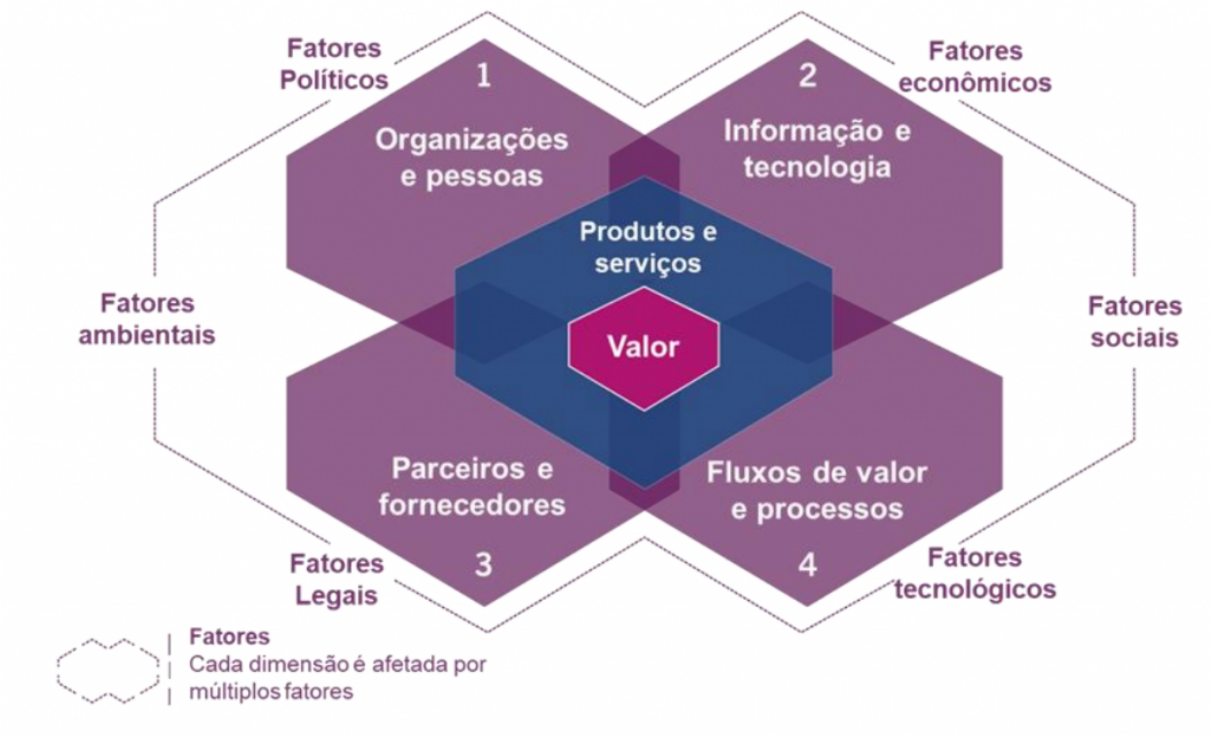

Figura 6 – Quatro dimensões do gerenciamento de serviços. (Fonte [4])

2.5.3 Gerenciamento e Desenvolvimento de *Software*

O ITIL4, adota algumas práticas para gerenciar serviços de TI. Uma prática de gestão é um conjunto de recursos organizacionais projetados para realizar um trabalho ou alcançar um objetivo. O ITIL4 inclui 34 práticas de gerenciamento. Para cada prática, há vários tipos de orientação, como termos e conceitos-chave, fatores de sucesso, atividades-chave ou objetos de informação. As práticas são divididas em três categorias, sendo elas [8]

1. Práticas gerais de gerenciamento
2. Práticas de gerenciamento de serviço
3. Práticas de gerenciamento técnico

No presente trabalho, focaremos na prática de Gerenciamento e Desenvolvimento de *Software*, cujo propósito é assegurar que os aplicativos atendam às necessidades das

partes interessadas internas e externas. Os aplicativos de *software*, desenvolvidos internamente ou em parceria com um fornecedor, desempenham um papel crucial na entrega de valor aos clientes em negócios orientados por serviços tecnológicos. Por consequência, o desenvolvimento e gerenciamento de *software* é uma prática chave em qualquer organização de TI moderna, garantindo adequação dos aplicativos para suas finalidades e usos. A prática de desenvolvimento e gerenciamento de *software* envolve atividades como: [8]

- Arquitetura da solução.
- Design da solução (incluindo interface do usuário, experiência do cliente, design de serviço, etc.).
- Desenvolvimento de *software*.
- Teste de *software* (abrangendo testes de unidade, integração, regressão, segurança da informação e aceitação).
- Gerenciamento de repositórios de código ou bibliotecas para manter a integridade dos artefatos.
- Criação de pacotes para eficiente implantação dos aplicativos.
- Controle de versão, compartilhamento e contínuo gerenciamento de blocos menores de código.

Ainda segundo [8], os dois modelos de desenvolvimento aceitos para o desenvolvimento de *software* são referidas como modelo prescrito e ágil. O gerenciamento de *software* é uma prática mais ampla, abrangendo as atividades contínuas de projetar, testar, operar e melhorar aplicativos de *software* para que continuem para facilitar a criação de valor. Os componentes de *software* podem ser continuamente avaliados usando um ciclo de vida que rastreia o componente desde a concepção até o contínuo melhora e, eventualmente, aposentadoria.

A qualidade de *software* é utilizada para descrever o *software* como produto e em seu uso, comumente em termos como: [34]

- qualidade do produto: adequação funcional, eficiência de desempenho, compatibilidade, usabilidade, confiabilidade, segurança, manutenibilidade e portabilidade;
- qualidade em uso: eficácia, eficiência, satisfação, ausência de riscos e cobertura de contexto.

Muitos dos requisitos para essas características de qualidade de *software* são insumos para o desenvolvimento e a gestão de software. Eles são determinados pelo proprietário do *software*. O componentes mais importantes para o fator de sucesso desta prática são: [34]

- compreender o código-fonte, como os vários módulos estão inter-relacionados e a arquitetura da aplicação;
- compreender os requisitos e o contexto em que a aplicação é utilizada;
- criar testes antes da codificação;
- controle efetivo de versão de todos os artefatos da aplicação;
- abordar a tarefa de codificação com plena compreensão de sua tremenda complexidade e respeitar as limitações intrínsecas da mente humana;
- adotar convenções de codificação;
- revisão por pares;
- *feedback* rápido proveniente dos testes, por exemplo, através do uso de testes automatizados, e tomar medidas corretivas rapidamente.

REFERÊNCIAS

[1] MAXIM, R. S. P. e B. R. *Engenharia de Software: uma abordagem profissional*. 9. ed. [S.l.]: AMGH, 2021.

[2] GOMES, C. *Scrum: A Metodologia Ágil Simplificada*. 2017. Disponível em: <https://blog.europneumaq.com/scrum-metodologia-agil-simplificada>. Acesso em: 21.10.2023.

[3] PRÁTICA, I. N. *ITIL®: o que é, para que serve e como tirar a certificação*. 2022. Disponível em: <https://www.itsmnapratica.com.br/tudo-sobre-itil/>. Acesso em: 31.10.2023.

[4] CéSAR, F. *ITIL 4: saiba tudo sobre a quarta versão do framework ITIL*. 2019. Disponível em: <https://www.profissionaisti.com.br/o-que-podemos-ganhar-com-itil-v4/>. Acesso em: 03.11.2023.

[5] HAMDAN, S.; ALRAMOUNI, S. A quality framework for software continuous integration. *Journal of Systems and Software*, v. 3, 2015.

[6] TELES, V. M. *Extreme Programming*. 2. ed. [S.l.]: Novatec, 2014. ISBN 9788575224007.

[7] FOSE 2014: Proceedings of the on Future of Software Engineering. New York, NY, USA: Association for Computing Machinery, 2014. ISBN 9781450328654.

[8] AXELOS. *ITIL Foundation - ITIL 4 Edition*. 4. ed. [S.l.]: Stationery Office, 2019. ISBN 9780113316076.

[9] FERNANDES, A. A.; ABREU, V. F. de. *Implantando a Governança de TI (4ª edição): da Estratégia à Gestão de Processos e Serviços*. 4. ed. [S.l.]: Brasport, 2014. ISBN 9788574526584.

[10] WOHLIN, C.; ŠMITE, D.; MOE, N. B. A general theory of software engineering: Balancing human, social and organizational capitals. *Journal of Systems and Software*, v. 109, 2015.

[11] SANCHEZ-GORDON, M.-L. et al. A standard-based framework to integrate software work in small settings. *Computer Standards & Interfaces*, v. 54, 2016.

[12] AL., B. et. *Manifesto for agile software development*. 2021. <https://agilemanifesto.org/>. Acessado em Outubro de 2023.

[13] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário: tradução*. 2. ed. Rio de Janeiro: Elsevier Brasil, 2006. ISBN 8535217843.

[14] SOMMERVILLE, I. *Engenharia de software*. 9. ed. São Paulo: Pearson Education do Brasil, 2011.

[15] WAZLAWICK, R. *Engenharia de Software - Conceitos e Práticas*. 2. ed. [S.l.]: GEN LTC, 2019. ISBN 9788535292725.

[16] BASTOS, A. et al. *Base de Conhecimento em Teste de Software*. 3. ed. São Paulo: Martins Fontes, 2007. ISBN 9788580630534.

[17] TIAN, J. *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. [S.l.]: Wiley, 2005. ISBN 9780471722335.

[18] GARCíA, F. et al. A framework for gamification in software engineering. *Journal of Systems and Software*, v. 132, 2017.

[19] ISO. *ISO/IEC 25000:2014*. 2014. Disponível em: <https://www.iso.org/standard/64764.html>. Acesso em: 09.10.2023.

[20] ISO. *ISO/IEC TR 25060:2010*. 2010. Disponível em: <https://www.iso.org/standard/35786.html>. Acesso em: 09.10.2023.

[21] SOARES, M. dos S. Comparação entre metodologias ágeis e tradicionais para o desenvolvimento de software. *INFOCOMP Journal of Computer Science*, v. 3, 2004.

[22] BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. 3. ed. [S.l.]: GEN LTC, 2014. ISBN 9788535226263.

[23] MEDEIROS, H. *Introdução ao Modelo Cascata*. 2017. Disponível em: <https://www.devmedia.com.br/introducao-ao-modelo-cascata/29843>. Acesso em: 11.10.2023.

[24] ALMEIDA, G. A. M. de. *Fatores de escolha entre metodologias de desenvolvimento de software tradicionais e ágeis*. Tese (Doutorado) — Universidade de São Paulo (USP), 2017.

[25] LAPLANTE, P. A. *What Every Engineer Should Know about Software Engineering*. Boca Raton, Florida: CRC Press, 2007. ISBN 9780849372285.

[26] SCHWABER, K. *Agile Project Management with Scrum*. Redmond, Washington: Microsoft Press, 2004. ISBN 9780735619937.

[27] SCHWABER, K.; SUTHERLAND, J. *The 2020 Scrum Guide ™*. 2020. Disponível em: <https://scrumguides.org/scrum-guide.html>. Acesso em: 18.10.2023.

[28] SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. Pennsylvania State University: Prentice Hall, 2002. ISBN 9780130676344.

[29] LEI, H. et al. A statistical analysis of the effects of scrum and kanban on software development projects. *Robotics and Computer-Integrated Manufacturing*, v. 43, p. 59–67, 2017.

[30] SHRIVASTAVA, A. et al. A systematic review on extreme programming. *Journal of Physics: Conference Series*, IOP Publishing, v. 1969, 2021.

[31] WELLS, D. *The Values of Extreme Programming*. 2009. Disponível em: <http://www.extremeprogramming.org/values.html>. Acesso em: 23.10.2023.

[32] MANSUR, R. *Governança de TI: Metodologias, Frameworks e Melhores Práticas*. Rio de Janeiro: Brasport, 2007. ISBN 9788574523224.

[33] PINHEIRO, I. L. M. e W. B. *Gerenciamento de serviços de TI na prática: uma abordagem com base na ITIL : inclui ISO/IEC 20.000 e IT Flex*. São Paulo: Novatec Editora, 2007. ISBN 9788575221068.

[34] AXELOS. *Software development and management: ITIL 4 Practice Guide*. 2020. Disponível em: <https://www.axelos.com/resource-hub/practice/software-development-and-management-itil-4-practice>. Acesso em: 10.11.2023.