INSTITUTO FEDERAL
SANTA CATARINA

CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE SAÚDE E SERVIÇOS
CURSO SUPERIOR DE TECNOLOGIA EM GESTÃO DA TECNOLOGIA DA INFORMAÇÃO

Lucas José Carvalho Teixeira

Engenharia de Software: análise de topologias para equipes de desenvolvimento

Florianópolis - SC
2022

Ficha de identificação da obra elaborada pelo autor.

Teixeira, Lucas
Engenharia de Software: análise de topologias para definições de equipes de desenvolvimento / Lucas Teixeira; orientação de Felipe Cantório Soares; coorientação de Egon Sewald Junior. – Florianópolis, SC, 2022.
59 p.

Trabalho de Conclusão de Curso (TCC) – Instituto Federal de Santa Catarina, Câmpus Florianópolis. CST em Gestão da Tecnologia da Informação. Departamento Acadêmico de Saúde e Serviços.
Inclui Referências.

1. Engenharia de software. 2. Equipes de desenvolvimento. 3. Topologias. 4. Lei de Conway. I. Cantório Soares, Felipe. II. Sewald Junior, Egon. III. Instituto Federal de Santa Catarina. IV. Engenharia de Software.

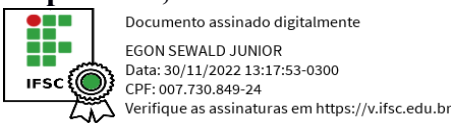
Engenharia de Software

Análise de topologias para definições de equipes de desenvolvimento

Lucas José Carvalho Teixeira

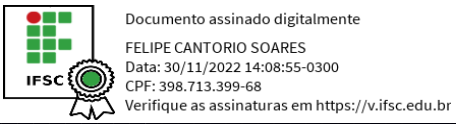
Este trabalho foi julgado adequado para obtenção do Título de Tecnólogo em Gestão da Tecnologia da Informação e aprovado na sua forma final pela banca examinadora do Curso Superior de Tecnologia em Gestão da Tecnologia da Informação do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis-SC, 22 de Novembro de 2022

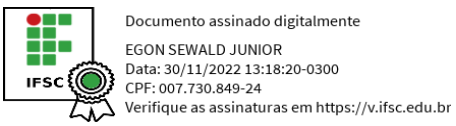


Prof. Egon Sewald Junior
Coordenador do CST em Gestão da Tecnologia da Informação
Instituto Federal de Santa Catarina

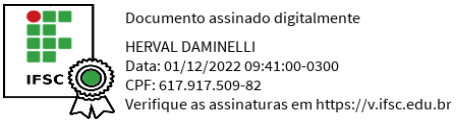
Banca Examinadora:



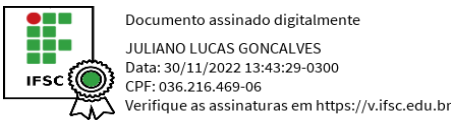
Prof. Felipe Cantório Soares
Orientador
Instituto Federal de Santa Catarina



Prof. Egon Sewald Junior
Coorientador
Instituto Federal de Santa Catarina



Prof. Herval Daminelli
Instituto Federal de Santa Catarina



Prof. Juliano Lucas Gonçalves
Instituto Federal de Santa Catarina

À Marina Giaquinto e a todas as pessoas que estão por aqui e as que pegaram o avião para um lugar melhor, em especial à Carmem.

AGRADECIMENTOS

Agradecimentos à Marina, companheira de vida, aos meus pais que me educaram e ainda me ensinam, ao Gael, Toya, Gabriel, Stella e Teka. Agradeço a todos os professores que me apresentaram a Gestão da Tecnologia da Informação, aos colegas de turma, aos meus orientadores Felipe Cantorio Soares, Egon Sewald Junior e aos que contribuíram na banca: Herval Daminelli e Juliano Lucas Gonçalves.

RESUMO

O presente trabalho, busca dentro da engenharia de software situar os conceitos que amparam a construção de equipes de desenvolvimento de software. Apresenta a chamada Lei de Conway e seus desdobramentos, que enfatizam a importância que a construção destas equipes sejam efetivadas mediante topologias que possuem um propósito e modos de interação que impactam na construção do sistema. Frente a isto, o trabalho busca também discutir como é a formação e interação de equipes de desenvolvimento no contexto do trabalho remoto que se consolidou a partir da pandemia do COVID-19.

Palavras-chave: Engenharia de Software, Equipes de Desenvolvimento, Topologias, Lei de Conway

ABSTRACT

The present work is about how the development teams are built. For this, concepts from software engineering will be used and as this will support teams topologies design. Also, going to present the Conway Law and your ramifications, that effort the importance of concepts to software development teams formation through topologies. Communication and interaction forms will be worked on along the text. Finally, in the remote work context during COVID-19, new challenges are posed to the software development teams and we will see the development teams can be built in this new context.

Key-words: Software Engineering, Development Teams, Topologies, Conway's Law

LISTA DE ILUSTRAÇÕES

Figura 1 - Modelo de desenvolvimento code-and-fix……..…… 15

Figura 2 - Topologia do modelo de equipe cirúrgica…….. 20

Figura 3 - Org Chart with Actual Line of Communication……...……...25

Figura 4 - Lei de Conway Reversa - Front-end e Back-end……...……..26

Figura 5 - Lei de Conway Reversa - Microsserviços……...……27

Figura 6 - Participantes das equipes de desenvolvimento……...…….29

Figura 7 - Número de Dunbar……..……...31

Figura 8 - Quatro tipo de topologias e como interagem……...35

Figura 9 - Níveis do trabalho remoto……..41

Figura 10 - Quadro Kanban……..44

SUMÁRIO

1. INTRODUÇÃO **10**

1.1 JUSTIFICATIVA **10**

2. DEFINIÇÃO DO PROBLEMA **12**

3. OBJETIVOS

3.1 Objetivo Geral 13

3.2 Objetivos Específicos 13

4. ESTRUTURA DO TRABALHO **13**

5. REVISÃO DA LITERATURA **14**

5.1 Engenharia de software e a organização de equipes 14

5.2 Times e topologias - Perspectiva sociotécnica. **22**

5.3 Como as equipes se formam 27

5.4 Quatro tipos de topologias 34

5.5 Organização de times de desenvolvimento no contexto remote-first 39

5.6 Identificar e reduzir as dependências (desacoplar as equipes) 43

6. PROCEDIMENTOS METODOLÓGICOS **48**

6.1 Caracterização da Pesquisa 48

6.2 Delimitação e População da Pesquisa 49

6.3 Etapas da Pesquisa 50

7. DESENVOLVIMENTO DO TRABALHO, APRESENTAÇÃO DE RESULTADOS **50**

8. CONCLUSÃO **57**

REFERÊNCIAS **58**

1 INTRODUÇÃO

No final dos anos de 1960, como uma resposta para o período que ficou conhecido como a crise do software, nasceu a Engenharia de Software (VALENTE, 2020). Seu objetivo foi organizar o processo de desenvolvimento de software de modo que mitigasse problemas como extrapolar orçamentos e prazos. Este movimento resultou na separação de assuntos e áreas que atravessam o processo de construção de sistemas, entre estas, a formação de times de desenvolvimento.

Os times de desenvolvimento têm se organizado diante de uma perspectiva operacional (FLORES, 2013), isto é, cada indivíduo tem sua habilidade técnica e a aplica no seu contexto de entrega de determinado projeto. No entanto, algumas questões podem adensar o entendimento sobre desenho de times de desenvolvimento de software.

Frente a esta discussão, sistematizar times, leva-se em consideração 1) a comunicação estabelecida entre os membros; 2) o impacto da arquitetura do software sobre o time; 3) o impacto da estrutura organizacional sobre a arquitetura do software e sobre a organização da equipe de desenvolvimento e 4) aspectos comportamentais e a carga cognitiva relacionada ao projeto.

1.1 JUSTIFICATIVA

O curso de Gestão de Tecnologia da Informação do IFSC, campus de Florianópolis, possui sua grade curricular distribuída entre disciplinas teóricas e técnicas. Neste sentido, ele resulta na amplitude da compreensão sobre o conceito de tecnologia, que vai além dos aspectos técnicos e por isso permite nos prepararmos para o engajamento em discussões conceituais.

O tema deste trabalho de conclusão de curso, é resultante dos anos de formação em Gestão da Tecnologia da Informação, que nos permite articular as pontas de assuntos tecnológicos com os aspectos conceituais que lhes estão atrelados, como times de desenvolvimento, interação e a produção de sistemas.

No contexto da atuação do gestor de tecnologia da informação, entender os conceitos que norteiam a formação de times, adensa a convicção que lhe é atribuída no seu espaço de atuação. Não se trata de conhecer apenas a necessidade técnica de um projeto e contratar o profissional com a habilidade para tal, mas sim manter o time em conformidade com a

arquitetura do software e construir um ambiente colaborativo. Para tal, faz-se necessário entender os conceitos que norteiam o design de equipes de desenvolvimento.

Compreendida a necessidade de ampliar o entendimento sobre a formação de times, autores como Skelton e Pais (2019) trazem modelos de tipologias que estão alinhadas com arquitetura, qualidade e performance que são aspectos demandados pelo atual mercado de trabalho. Esses aspectos são resultantes da crescente demanda por software, que são suportados por novas arquiteturas, novos recursos computacionais que geram inovação e impactam na maneira de interagir e organizar times.

Entre as grandes inovações que impactam o mercado de tecnologia da informação, o setor de *cloud computer* tem um crescimento notável[1]. A arquitetura hospedada em nuvem se adequa à arquitetura de microsserviços (Fowler, 2020), que por sua vez, exige uma compreensão distinta sobre a formação de times, cultura organizacional e papel de cada membro do time.

Microsserviços dizem respeito à arquitetura do sistema, visando um sistema segmentado em partes menores, escalável (que tenha a capacidade de crescer e suportar aumento de carga), eficiente e que colabore na velocidade do desenvolvimento de software. Esse padrão de arquitetura é uma solução para sistemas tidos como monolitos. Softwares com arquitetura acoplada, não segmentado que possui dificuldade para crescer, expandir seu ciclo de vida quando recebe maior carga de trabalho (Fowler, 2020).

> O conceito básico de um microsserviço é simples: ele é uma pequena aplicação que executa uma única tarefa e o faz com eficiência. Um microsserviço é um pequeno componente que pode ser facilmente substituído, e é desenvolvido e implantado de forma independente. Um microsserviço não sobrevive sozinho, porém - nenhum microsserviço é uma ilha -, é parte de um sistema maior sendo executado e funcionando junto a outros microsserviços para realizar tarefas que normalmente seriam tratadas por uma grande aplicação autônoma (Fowler, 2021, p 26).

Portanto, para acompanhar o ritmo do mercado e manter a velocidade e qualidade da entrega e manutenção dos softwares, faz-se necessário entender os diferentes conceitos que perpassam pela construção de times e menos a ação baseada em *feeling*.

---

[1] Segundo reportagem da Inmetrics: “Fornecedoras de soluções em cloud, como a Amazon Web Services (AWS), Google Cloud e Microsoft Azure aumentaram suas demandas nos primeiros meses da pandemia. Segundo pesquisa do Synergy Research Group, houve um aumento de 37% no investimento em cloud nos primeiros quatro meses de 2020.”**. Disponível em: <https://inmetrics.com.br/blog/cloud-computing-como-a-pandemia-acelerou-a-implementacao/> Acesso em 05/2021.**

1.2.DEFINIÇÃO DO PROBLEMA

Como colocado por Flores (2013), aspectos conceituais importantes para a formação de equipes de desenvolvimento, acabam sendo escanteados, pela noção de feeling, experiência e outras medidas no âmbito pessoal de gestores.

O crescimento da demanda por software, consequentemente aumentou a demanda de profissionais qualificados para ocupar as vagas de emprego. Esse crescimento adensou durante a pandemia. Como a oferta de trabalho é mais alta do que a disponibilidade de pessoas no mercado para ocupar as vagas de trabalho, as empresas buscam ser atrativas tornando a rotatividade dentro dos times de desenvolvimento alta. Por outro lado, a demanda por software permanece crescente.

Seguem abaixo alguns dados do atual patamar do mercado de tecnologia da informação, extraídos entre os anos de 2019 e 2020 pela Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação (BRASSCOM)[2]. Em 2020 foram gerados 43.624 postos de empregos em tecnologia da informação. Nos dados do ano seguinte, extraídos em setembro de 2021, o número saltou para 123.544 novos empregos.

Em comparação com o ano de 2020, houve o crescimento de 183%. Até o ano de 2025, há a projeção para o intervalo de 2021 até 2025, é a necessidade de 797 mil trabalhadores para ocupar as áreas de *Big Data e Analytics*, *Nuvem* e *Web Mobile.*

Outra variável que devemos levar em conta, é como que a partir de 2020 o trabalho remoto passou a ser comum a partir da pandemia da COVID-19. Criou-se uma nova marca, chamada de *remote first*. Esse modelo impactou através de novas maneiras de comunicação e relacionamento entre a equipe de desenvolvimento. Os times distribuídos também tornaram-se pauta da engenharia de software.

A definição de remote-first, que em tradução livre seria remoto primeiro, é a seguinte:

> … é uma estratégia organizacional que faz do trabalho remoto a primeira opção para muitos ou todos os empregados. Por definição, remote-first significa que poucas pessoas são exigidas para trabalharem em escritórios centralizados. Ao invés disso, as pessoas trabalham em outras localizações, como em casa ou em espaços de coworking[3].

---

[2] Para mais detalhes, segue o relatório completo: BRASSCOM. **Demanda de talentos em TIC e Estratégia** . Relatório de Inteligência e Informação BRI2-2021-007 - v112. Disponível em: <https://brasscom.org.br/demanda-de-talentos-em-tic-e-estrategia-%cf%83-tcem/>. Acessado em 07/2022.

[3] **What is remote-first?**"Remote-first" is an organizational strategy that makes working remotely the primary option for most or all employees. By definition, remote-first means that few (if any) people are regularly required to perform their jobs from a centralized office. Employees instead work from another location, such as a home office or co-working space. Disponível em:

Diante deste cenário, o presente trabalho se propõe a retomar a discussão: como os modelos de topologia de times de desenvolvimento impactam na construção do software e, por sua vez, como a arquitetura do software impacta sobre esses times.

3 OBJETIVOS

3.1 Objetivo Geral

Analisar quais são os conceitos que devem nortear a formação de equipes de desenvolvimento.

3.2 Objetivos Específicos

- Listar quais são os conceitos que norteiam as diferentes topologias de times;
- Examinar se os times são organizados de maneira reativa e respondem conforme demandas ou se são racionalizados;
- Descrever como as arquiteturas de sistemas refletem sobre a topologia dos times;
- Descrever como a organização dos times refletem sobre a arquitetura dos sistemas.

4. ESTRUTURA DO TRABALHO

O presente trabalho é composto por cinco capítulos. O primeiro é a Introdução, nele discutimos a motivação e a relevância do tema dentro da engenharia de software e sua importância no contexto profissional da gestão da tecnologia da informação. Ainda neste capítulo, traçamos o objetivo geral e os específicos da pesquisa focado na abordagem que relaciona interação entre a equipe de desenvolvimento mais a estrutura organizacional com o sistema produzido.

No segundo capítulo, temos a Revisão da literatura. Nele, todas as referências lidas e os conceitos extraídos que compõem o texto, estão focados no objetivo geral e nos

---

<https://www.vmware.com/topics/glossary/content/remote-first.html?resource=cat-803692625#cat-803692625> Acessado 09/2022.

específicos, sobre quais são os conceitos que devem fazer parte na formação de times de desenvolvimento de software.

No terceiro capítulo, descrevemos os procedimentos metodológicos que balizaram a construção da pesquisa.

Em penúltimo, o quarto capítulo, são apresentados os resultados deste trabalho frente ao que foi proposto nos objetivos.

Por fim, o último capítulo aborda a conclusão da pesquisa.

5. REVISÃO DA LITERATURA

5.1 Engenharia de software e a organização de equipes

A engenharia de software nasceu com o objetivo de sistematizar processos, metodologias, estrutura organizacional e fornecer isto como recursos para construção racionalizada de softwares. Seus diversos setores, concentram-se em processos organizacionais, modelos de desenvolvimento, padrões de projetos, arquitetura de software, testes, versionamento, levantamento de requisitos, dentre outros.

Em suma, engenharia de software, "preocupa-se em estudar e monitorar o bom andamento de todas essas atividades e a integração entre elas, é trivial notar que é baseado na Engenharia de Software o sucesso de um projeto no que tange a sua organização." (MORAIS, Izabelly Soares D.; ZANIN, Aline, 2020, p 17).

Assim como outras áreas da computação, que nasceram no ambiente acadêmico e militar, o mesmo se aplica à Engenharia de Software, nascida no fim dos anos de 1960.

> A criação de software foi subestimada e realizada sem nenhuma metodologia, gerando erros em sistemas, como problemas de cálculos e perdas financeiras e de tempo. Nesse período, podemos dizer que houve a crise do software. Com isso, em 1968, a Organização do Tratado do Atlântico Norte (OTAN) designou o termo Engenharia de Software para adequar o processo de desenvolvimento de software com metodologias já utilizadas em outras Engenharias. Uma série de metodologias e técnicas passaram a ser utilizadas antes, durante e depois da criação de software (MORAIS, Izabelly Soares D. ZANIN, Aline, 2020, p 12).

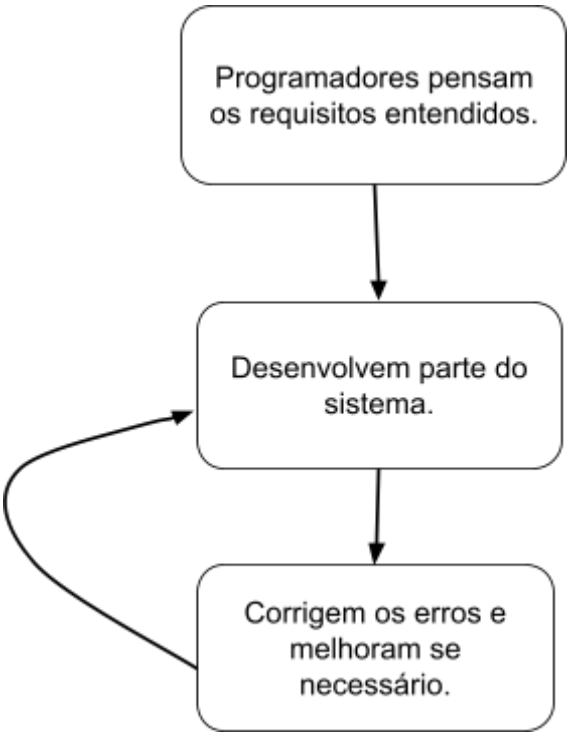
No livro Engenharia de Software Moderna de Marco Túlio Valente publicado em 2020, o autor traz uma fala importante de um dos participantes da Conferência da OTAN de 1968. Neste trecho, temos uma expressão da *crise do software.*

> O problema é que certas classes de sistemas estão colocando demandas sobre nós que estão além das nossas capacidades e das teorias e métodos de projeto que conhecemos no presente tempo. Em algumas aplicações não existe uma crise, como rotinas de ordenação e folhas de pagamento, por exemplo. Porém, estamos tendo dificuldades com grandes aplicações. Não podemos esperar que a produção de tais sistemas seja fácil . (Valente. M. T 2020).

Para contextualizar o momento da crise do software, vejamos mais elementos característicos anteriores à Engenharia de software. Entre os anos de 1960 e 1970, havia uma limitação em que não se encaixava aplicar o mesmo processo e técnicas de desenvolvimento de pequenos programas em projetos mais amplos e complexos. No texto *The software crisis,* produzido em 2011 pelo departamento de ciência da computação da University of Cape Town, descreve um modelo típico da época chamado *code-and-fix*.

Code-and-fix, como mostra a tradução, era a prática de desenvolvedores reunidos com uma lista de requerimentos vagos e a partir destes, desenvolviam e corrigiam sucessivamente, até ficarem satisfeitos com o resultado. Este modelo, em escala menor e com sistemas com menos complexidade com ressalvas, funcionava, porém, se demonstrou insustentável quando aplicado a projetos de sistemas maiores e mais complexos.

Figura 1 - Modelo de desenvolvimento code-and-fix



Fonte: Notas do MIT sobre engenharia de software[4]

[4] Disponível em: <https://www.cs.uct.ac.za/mit_notes/software/htmls/ch02s02.html>. Acesso em: maio 2022.

Entre os problemas que remontam a crise do software estão: os sistemas nunca se completavam, havia falta de prazos, o orçamento excedia, o sistema não cumpria com todos os requisitos, sistema que era difícil de se usar, dificuldade para realizar mudanças que atendiam as necessidades organizacionais, perda de confiança dos usuários, uma vez que muitos problemas ocorriam durante o uso[5].

Resumidamente, como demonstra NUNES (2016), a engenharia de software, no fim dos anos de 1960, foi uma reação para um contexto que ficou marcado pela crise na criação de softwares. Neste momento, aumentava a demanda e complexidade dos projetos, porém, ainda não havia princípios norteadores sólidos no processo de desenvolvimento, tornando assim a Engenharia de Software, como:

> (...) uma resposta à chamada crise de software que surgiu com as experiências associadas ao projeto, desenvolvimento e manutenção de sistemas de software confiáveis de larga escala. Os sistemas de software eram construídos como torres de Babel: sem planejamento, sem comunicação, sem documentação etc. Muitos projetos foram abortados e a história tem inúmeros casos de prejuízos causados (NUNES, D. J. 2015, p.17).

Dentre os tópicos da Engenharia de Software, destacamos o foco do trabalho sobre como os modelos de topologia de times de desenvolvimento impactam na entrega do software e como são impactados pela organização em que estão inseridos.

Quais são os conceitos que norteiam as diferentes topologias de times? Os times são compostos e ordenados de maneira racionalizada ou respondem conforme o volume de demandas?

Em 1975, Frederick P. Brooks, publicou o livro de ensaios *O Mítico Homem Mês*. É considerado a primeira obra de maior relevância da Engenharia de Software que conseguiu reunir ensaios baseados na experiência do autor como gerente do projeto OS/360 em 1965. O livro preconiza alguns pontos que são do interesse do tema da pesquisa, os quais passaremos a seguir.

Um dos principais ensaios do O *Mítico Homem Mês,* o autor descreve que em desenvolvimento de software, pessoas não são intercambiáveis, ou seja, não basta trocar uma pessoa por outra e esperar a normalidade da entrega. No contexto em que projetos de sistemas

[5] Texto original: never completed systems; missed deadlines; exceeded budgets; a system that does not do all that is required of it; a system that works but is difficult to use; a system difficult to modify to meet changes in organizational needs and practices; a loss of trust from users, who may experience many problems with using the software.

podem ultrapassar a estimativa inicial equivocada, a tendência da gerência é acrescentar mais *homem-mês*, como um movimento reativo para tentar cumprir o prazo de entrega.

Segundo o que ficou conhecido como lei de Brooks:

> Como a construção de software é, por natureza, um trabalho sistemático - um exercício de inter-relações complexas - o esforço de comunicação é grande e rapidamente domina a diminuição do tempo de cada tarefa individual que foi estabelecido na sua divisão. A adição de mais homens, portanto, aumenta, e não diminui o cronograma (BROOKS, 2018, P).

No trecho acima, há um reconhecimento sobre a comunicação entre as pessoas da equipe de desenvolvimento como uma característica inerente ao trabalho de construção de software. E para o autor, isso deve ser levado em consideração na estimativa do projeto. É também um reconhecimento importante para levar em consideração na composição de times de desenvolvimento, a comunicação entre os membros da equipe.

As topologias dos times de desenvolvimento, quando cientes da sua característica inter-relacional, buscam abrir rotas e criar canais para propor a comunicação entre pessoas do time e, ao mesmo tempo, mitigar erros por falhas comunicativas.

Para Frederick P. Brooks, o propósito de um modelo organizado no desenvolvimento de software, é garantir aquilo que ele chamou de *integridade conceitual.* A integridade conceitual, segundo o autor, é ponto alto de um projeto de desenvolvimento, ela é responsável por manter as ideias concisas, não ambíguas e/ou desencontradas, que resultem em programas simples e diretos (BROOKS, 2018).

Considerando que Brooks escreveu antes do lançamento das metodologias ágeis, era comum atrasos de meses e até em anos para conclusão de projetos longos. No decorrer destes projetos de software, o rodízio de “projetista” e a quebra em múltiplas subtarefas executadas por muitas pessoas, resultava no comprometimento da integridade conceitual.

> Vou sustentar que a integridade conceitual é o ponto mais importante no projeto de sistemas. É melhor que um sistema omita certas funcionalidades anônimas e melhorias e reflita um conjunto de ideias de projeto, do que ter um projeto que contenha muitas boas ideias, mas independentes e desencontradas (BROOKS, 2018, P40.).

O contexto de desenvolvimento de software da sua época, década de 1970, eram comuns projetos com estimativas longas, cronogramas apertados e, principalmente, muitas pessoas envolvidas. Diante disto, o autor sugere dois modelos que poderiam preservar a integridade conceitual: a *equipe cirúrgica* e a divisão entre *arquitetura e implementação.*

O conceito de *equipe cirúrgica* em desenvolvimento de software, foi apresentado por Harlan Mills presente em dois artigos de 1968 e outro de 1971 e, em suma, diz respeito às vantagens da execução de um projeto por um número reduzido de pessoas. Um ponto interessante, é que essa ideia preconiza o modelo de organização de equipes na metodologia ágil, no conceito de *equipe* dentro do Scrum: " (...) a regra básica é que sete integrantes são o ideal - podendo ter mais ou menos dois. Eu diria menos" (Sutherland, 2019, p. 77).

O Scrum, motivado pelo Manifesto Ágil de 1991[6], que reorientou o modelo de equipes e processos de desenvolvimento de software como uma solução aos problemas resultantes do modelo cascata, predominante desde os anos de 1970, em que o processo de desenvolvimento era longo, custoso e burocrático. A entrega ao cliente no Scrum é feita sob ciclos curtos de desenvolvimento, equipes menores, que buscam ser eficientes.

A equipe cirúrgica por Brooks, é explicada com o exemplo de dez pessoas participantes da criação de software. Os atores são: o *cirurgião*, o *copiloto*, o *administrador*, o *editor*, dois *secretários*, o *escriturário de programação,* o *ferramenteiro*, o *testador* e, por fim, o *testador*. Veremos os respectivos papéis.

Vejamos em detalhes, a partir do livro do *Mítico Homem Mês(2018):*

Tabela 1 - Atores e funções dentro da equipe cirúrgica.

Ator	**Função**
Cirurgião	Define as funcionalidades, especificações de desempenho, arquitetura, codifica, testa e documenta, faz manutenção. Sua função requer talento, experiência e considerável conhecimento em sistemas e aplicações.
Copiloto	Conhece o sistema tanto quanto o cirurgião, porém tem menos experiência. Atua como um pensador, crítico e avaliador do sistema, representa sua equipe em discussões sobre funcionalidades e interfaces, buscando alternativas para o projeto.
Administrador	Lida com dinheiro, espaço, pessoas e máquinas. caso o projeto tenha assuntos burocráticos e legais, o recomendado é que este seja dedicado integralmente à equipe.
Editor	Responsável por escrever a documentação a partir do que

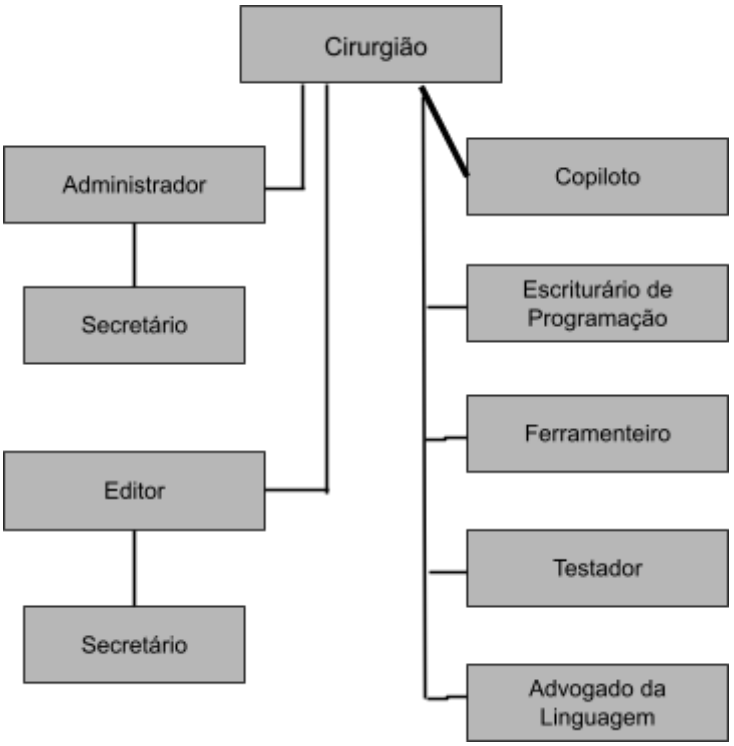
[6]Os pilares do Manifesto Ágil (1991) são: **Indivíduos e interações**, mais do que processos e ferramentas; **Software em funcionamento**, mais do que documentação abrangente; **Colaboração com o cliente**, mais do que negociação de contratos; **Resposta a mudanças**, mais do que seguir um plano.

	foi passado pelo cirurgião. Ele pode sugerir alterações, correções e cuidados como referências bibliográficas.
Dois secretários	Um secretário será designado para auxiliar o administrador e outro para o editor.
Escriturário de programação	É responsável por sistematizar os dados legíveis para máquinas e para humanos. Registra todos os dados de entrada e saída, disponibiliza todas as execuções do computador para toda a equipe.
Ferramenteiro	Seu papel é cuidar para que os recursos necessários para o trabalho do cirurgião estejam disponíveis de maneira confiável e satisfatória.
Testador	Planeja e escreve testes levando em consideração a especificação funcional e aquilo que foi entregue pelo cirurgião.
Advogado da Linguagem	Pessoa que tem o domínio sobre a linguagem utilizada no projeto, encontra maneiras simples de resolver problemas complexos, é atento às boas práticas.

Fonte: O Mítico Homem-Mês, 2018.

Para Brooks (2018), esse modelo com dez pessoas, entrega sistemas melhores porque consegue atingir a integridade conceitual do projeto, consequência de uma estrutura simples que facilita a comunicação entre o time. Segue abaixo a topologia apresentada por Brooks baseada no modelo de equipe cirúrgica.

Figura 2 - Topologia do modelo de equipe cirúrgica.



Fonte: Modelo extraído de O Mítico Homem-Mês 2018[7].

O modelo de *arquitetura e implementação,* é outra estrutura para atingir a integridade conceitual, porém voltada para projetos grandes (Brooks, 2018). Quais são os papéis do arquiteto e do implementor?

> Por arquitetura de um sistema, refiro-me à especificação completa e detalhada da interface do usuário, para um computador, ela é o manual de programação. Para um compilador, o manual da linguagem. Para um programa de controle, são os manuais para a linguagem ou as linguagens para chamar suas funções. Para um sistema completo, é a união de todos os manuais que o usuário deve consultar para executar integralmente suas funções (BROOKS, 2018, P44).

O autor complementa dizendo que o arquiteto é o representante do usuário, responsável por mobilizar todo seu conhecimento profissional e técnico para satisfazer as intenções do usuário, não de uma empresa que encomendou um software, mas do usuário final (BROOKS, 2018).

[7] BROOKS, Frederick Phillips. **O mítico homem-mês**: ensaios sobre engenharia de software. Tradução: César Brod. Rio de Janeiro: Elsevier, 2018.

Já o implementador, o autor diz que é o responsável por construir os componentes internos e tudo aquilo que faz o sistema que foi especificado pelo arquiteto. Seu papel não fica restrito à implementação funcional e técnica sem contato com a criatividade. Brooks, toma cuidado de não desnivelar a importância dos dois grupos para a construção de software.

Os implementadores, tendo o mínimo necessário de especificação, podem também começar seu trabalho projetando os fluxos de dados no sistema, conceber as ferramentas necessárias. Também podem verificar se há detalhes e excessos que podem atrapalhar a implementação e por isso serem postergados.

O resultado deste modelo é a entrega do sistema mais rapidamente, os testes são executados mais rápido, a comunicação ganha com simplicidade e melhora a integridade conceitual do produto.

É importante notarmos que Brooks (2018) enfatiza a questão da comunicação e organização de processos como fatores que tornam determinados projetos bem sucedidos nos primórdios da engenharia de software. E conforme veremos adiante, novas tipologias vão estruturar times em contextos de desenvolvimento de software contemporâneos, porém sem perder de vista a ênfase na organização e comunicação.

Para reforçar seu argumento sobre o papel primordial da comunicação, o autor usou como metáfora o mito da Torre de Babel no capítulo *Por que a Torre de Babel falhou?* A incapacidade de comunicação e consequentemente de organização foi sua resposta. O comprometimento da ação comunicativa gradativamente degrada um projeto. "... a falta de comunicação gera disputas, ressentimentos e ciúmes no grupo. Em pouco tempo, os clãs começaram a distanciar-se, preferindo o isolamento à disputa" (Brooks, 2018, p 72).

Ambos *modelos de equipe*, estão contextualizados dentro do *modelo de desenvolvimento de software waterfall,* que era o que havia disponível com a promessa de ser simples e funcional para sua época[8], antes do período de crise de software. Tinha por pressuposto que os sistemas poderiam ser inteiramente desenvolvidos em uma única vez. No entanto, problemas com performance baixa, problemas com usabilidade, eram descobertos apenas no fim, no momento dos testes (Brooks, 2018).

Levando em consideração, a lei de Brooks, os modelos de equipes pensadas pelo autor, o modelo de desenvolvimento de software da sua época, a questão central que está colocada na comunicação e, consequentemente, nas pessoas.

---

[8] Vinte anos após o lançamento do livro, Brooks escreveu sua crítica ao modelo waterfall, no capítulo *O Mítico homem mês, 20 anos depois.* O título do tópico é autoexplicativo: *Não Construa Um para Descartar Depois - O Modelo Waterfall Está Errado!* . Sua defesa está no título do tópico subsequente: *Um Modelo Incremental de Construção é Melhor - Refinamento Progressivo.*

Por fim, para o autor, independente da tipologia definida por uma organização, "… uma questão central que se apresenta ao gerente de software é o projeto de uma estrutura e processos que aumentem ao invés de inibir, a criatividade e a iniciativa”. (Brooks, 2018, p 267). Em outras palavras, o autor sugere a distribuição de poder como modelo para resultados bem sucedidos em times de desenvolvimento.

Podemos inferir que do último quarto do século XX, ano que Frederick P. Brooks, escreveu sua obra, até o século XXI, o crescimento da demanda por projetos de software foi exponencial com o crescimento da dependência da informática no âmbito das transações econômicas e a popularização dos recursos informatizados para automação de processos dentro do setor de serviços.

Esse crescimento se adensou com a popularização do comércio eletrônico e do uso de dispositivos móveis com acesso à internet. Por consequência, houve também aumento da demanda de projetos de software.

Dentro dos times de desenvolvimento, os ecos desse crescimento e transformação impactou na organização dos times.

5.2 Times e topologias - Perspectiva sociotécnica.

A perspectiva sociotécnica considera que uma organização é moldada pela interação entre indivíduos e tecnologia. No contexto de engenharia de software, esta interação está descrita, de certa forma, naquilo que falamos sobre humanos na perspectiva de Brooks.

Porém, podemos adensar no viés sociotécnico quando estudamos a Lei de Conway, introduzida em 1967 pelo cientista da computação Melvin E. Conway. Em linhas gerais, a Lei de Conway diz que o desenho de uma organização é projetado sobre a arquitetura do software. O sistema é uma cópia da estrutura de comunicação de sua organização.[9]

Para Melvin E. Conway, semelhante à Lei de Brooks, as organizações deveriam deixar de lado a suposição de que acrescentar mais força de trabalho resultaria no aumento da produtividade. O autor conclui que a organização precisa ter uma nova filosofia que apresente esclarecimento sobre os valores dos recursos e técnicas de comunicação antes de construir seu produto tecnológico.

A Lei de Conway está viva nas discussões sobre organização de times e topologias atualmente. No livro *Manual DevOps - Como obter agilidade, confiabilidade e segurança em*

[9] The basic thesis of this article is that organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations.

organizações tecnológicas de 2018, há um capítulo dedicado sobre o tema. Os autores descrevem o seguinte caso:

> À medida que as organizações crescem, um dos maiores desafios é manter comunicação e coordenação eficientes entre pessoas e equipes. Com muita frequência, quando pessoas e equipes ficam em pavimentos diferentes, em prédios diferentes ou em fuso horário diferente, fica mais difícil criar e manter um entendimento compartilhado e confiança mútua, impedindo a colaboração efetiva. A colaboração também é impedida quando os principais mecanismos de comunicação são ordens de serviço e pedidos de mudança, ou pior, quando as equipes estão separadas por limites contratuais, quando o trabalho é realizado por equipe terceirizada. (Vários Autores, 2018, p 89)

Por consequência, a arquitetura do software fica suscetível a sofrer as consequências negativas da *Lei de Conway*, podendo ter como produto com uma arquitetura muito acoplada. Por outro lado, a alternativa seria manter equipes menores com estrutura comunicativa reduzida e dedicada a um escopo específico. O esperado seriam entregas de arquiteturas menos acopladas e escaláveis, como ocorre em empresas como Amazon e Google mencionadas no capítulo pelos autores.

Em 2015, a Lei de Conway ressurgiu diante das discussões sobre arquitetura de microsserviços. Na ocasião, James Lewis, na época diretor técnico na empresa de Thoughtworks, entre outros, discutiram o que chamaram de Lei de Conway Reversa (SKELTON; PAÍS, 2019).

Como o próprio nome diz, Lei de Conway Reversa, parte do princípio reverso no qual a estrutura organizacional de uma empresa é ordenada pela arquitetura de seu software.

Interessante notar que a AWS (Amazon Web Services), traz na sua definição de microsserviços tanto seu arranjo organizacional quanto o técnico, vejamos:

> Microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes que se comunicam usando APIs bem definidas. Esses serviços pertencem a pequenas equipes autossuficientes (MICROSSERVIÇOS, 2022).

Segundo Fowler(2021), empresas em que a arquitetura de seus produtos estão organizadas como microsserviços, apresentam na prática a Lei de Conway reversa:

> A arquitetura de microsserviços consiste em um grande número de pequenos microsserviços independentes e isolados. A Lei de Conway Reversa exige que a estrutura organizacional de toda empresa que usa arquitetura de microsserviços seja composta de um grande número de equipes muito pequenas, isoladas e independentes. A estrutura de resultantes inevitavelmente leva aos fenômenos de segregação e dispersão, problemas que pioram sempre que o ecossistema se torna

> mais sofisticado, mas complexo , mais simultâneo e mais eficiente (Fowler, p. 47, 2021).

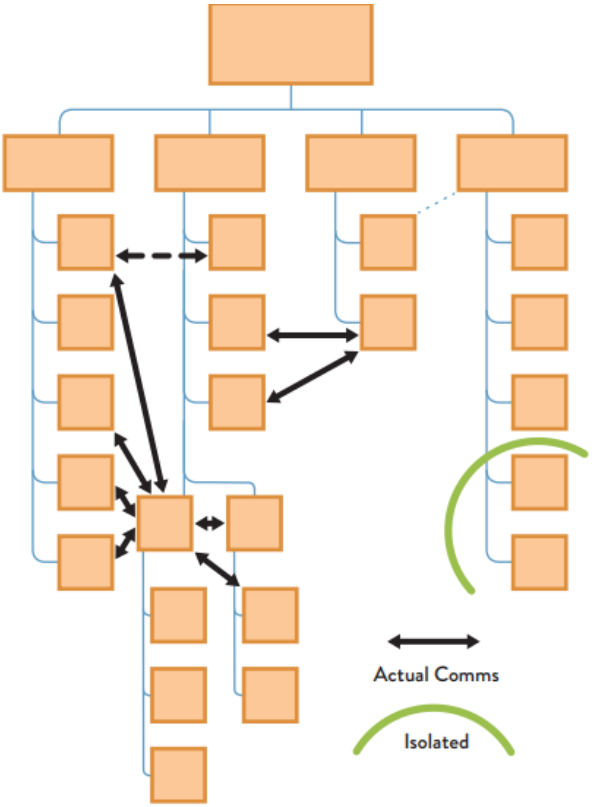
Segundo a autora, o modelo de microsserviços tende a criar um grau de individualidade e especialização sobre o sistema, isolando do alcance do desenvolvedor a visão sobre toda arquitetura do sistema.

Fowler diz que o desafio organizacional colocado para empresas que lidam com microsserviços, é trabalharem de maneira interativa entre si para manter o produto operando corretamente (Fowler, 2021). Frente a isto, podemos destacar novamente a importância da comunicação para Brooks como parte de inter-relações complexas e necessárias para integridade conceitual do software. E para Conway, a necessidade do esclarecimento da organização sobre os recursos e técnicas de comunicação que devem anteceder a construção de um produto.

Como solução de custos, escalabilidade, disponibilidade, facilidade na manutenção, algumas organizações utilizam serviços de nuvem para distribuir e organizar seu produto. No entanto, podemos fazer o seguinte questionamento, essas organizações estão atentas ao grau de especialização e dispersão que este modelo pode resultar? Consideram adequar seu organograma, maneiras de interação e técnicas de comunicação para este tipo de serviço?

No gráfico organizacional abaixo (Figura 3), temos um exemplo de estrutura rígida, com canais de comunicação escassos. Decisões baseadas neste contexto organizacional de comunicação *top-bottom,* possuem efeitos que melhoram para os que estão diretamente envolvidos nas decisões. Por outro lado, seus efeitos ignoram os que estão posicionados mais acima ou mais abaixo do quadro organizacional (SKELTON; PAIS, 2019).

Figura 3. Org Chart with Actual Line of Communication.



Fonte: Extraído do livro: Team Topologies[10]

Segundo Skelton e Pais (2019), a força da estrutura organizacional reflete sobre a arquitetura do sistema. Dando continuidade a esta correlação de forças, os autores também identificaram aquilo que chamaram como a versão moderna da Lei de Conway.

Atribuída à Ruth Malan, antiga diretora da Thoughtworks, a frase diz o seguinte: "se a arquitetura de um sistema e a arquitetura de uma organização estão em desacordo, a estrutura da organização vence" (SKELTON; PAIS, 2019 p 19). Segundo Ruth Malan, isso ocorre porque as organizações devem produzir projetos que imitem sua estrutura comunicacional (SKELTON; PAIS, 2019).

Para Matthews Skelton e Manuel Pais, autores que escreveram o livro *Team Topologies - Organizing Business and Technology Teams for Fast Flow*, organizações em que as decisões são baseadas no organograma com canais comunicativos falhos, não garantem entrega rápida e segura para seus clientes. Os autores dão o seguinte exemplo:

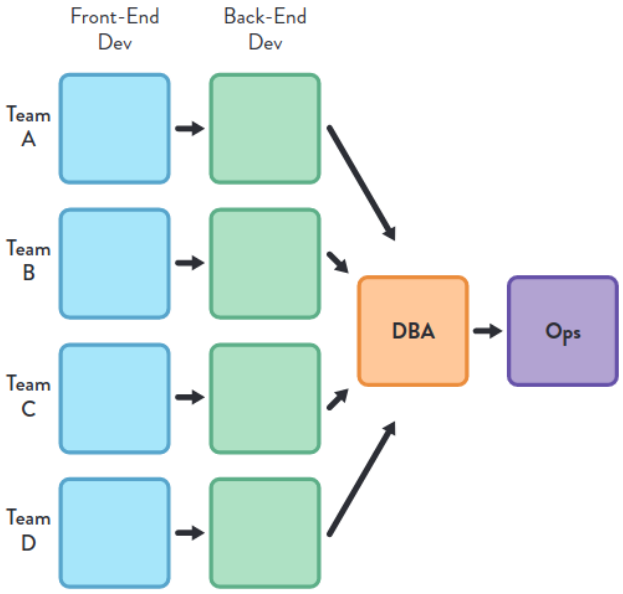
[10] SKELTON, Matthew; PAIS, Manuel. **Team Topologies**: organizing business and technology teams for fast flow. Portland: It Revolution, 2019. 216 p.

> (...) equipes adotam cloud computer e infraestrutura como código para poder reduzir o tempo de provisionamento de novas infraestruturas de meses, semanas para minutos. Mas se alguma mudança requer deploy (para o ambiente de produção), um quadro de aprovação se reunirá uma vez por semana, então a velocidade da entrega será semanal, na melhor das hipóteses (SKELTON; PAIS. 2019 p 5. Tradução nossa).

A Lei de Conway reversa estabelece uma prerrogativa importante. Recomenda-se que antes de definir a arquitetura do software, é importante que se organize a estrutura comunicativa da equipe. Neste sentido, é necessário que o time seja estruturado de acordo com os requisitos existentes na arquitetura do software (SKELTON; PAIS, 2019).

Na figura abaixo (Figura 4), temos uma topologia que demonstra quatro times. O time de desenvolvimento, está dividido entre front-end e back-end, ambos se comunicam. O banco de dados é compartilhado entre os desenvolvedores de backend, que se comunicam com os administradores de banco de dados que, por sua vez, se relacionam com o time de operações.

Figura 4. Lei de Conway Reversa - Front-end e Back-end.



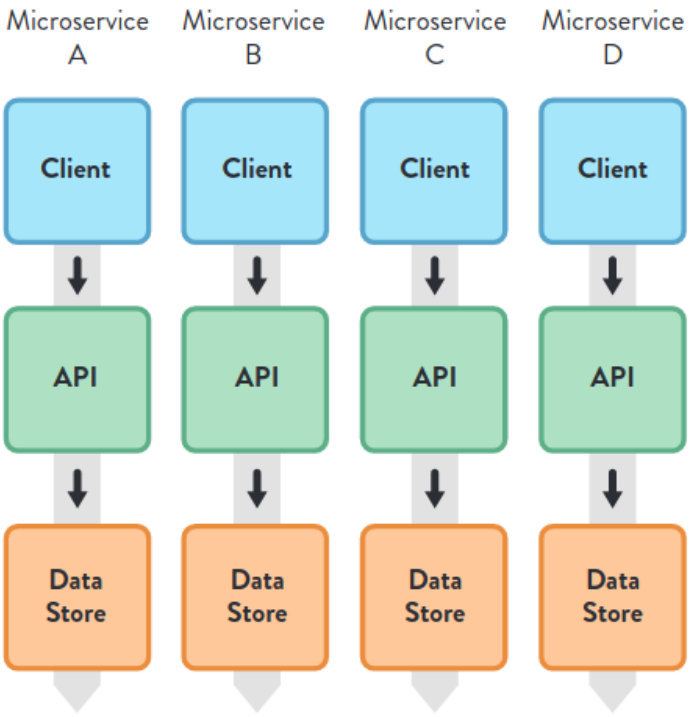
Fonte: Extraído de Team Topologies[11]

Este modelo, resulta da arquitetura de uma aplicação definida com responsabilidades de back-end e front-end separadas e com um banco de dados compartilhado entre a equipe.

No caso de uma topologia baseada no modelo de uma aplicação distribuída em microsserviços, a distribuição do time seria a seguinte:

[11] *Ibidem, p. 19*

Figura 5. Lei de Conway Reversa - Microsserviços



Fonte: livro Team Topologies[12]

Neste caso, os desenvolvedores são separados por cliente e lida com back-end e front-end respectivamente com os microsserviços distribuídos isoladamente.

5.3 Como as equipes se formam

Ainda seguindo a Lei de Conway e a interpretação da Lei de Conway Reversa, são duas chaves que orientam o modelo de construir a composição das equipes. Levando a sério como as escolhas organizacionais impactam na arquitetura de software que se almeja, bem como a estruturação dos canais comunicativos, a estruturação da equipe e definição da expertise técnica compõem o resultado final, que é a entrega do produto.

Segundo Skelton e Pais (2019), citando Ruth Malan (2008), “se nós temos gestores decidindo (...) que serviços serão construídos por qual time, nós temos implicitamente

[12] *Ibidem,* p. 21

gestores decidindo sobre a arquitetura do sistema" (SKELTON; PAIS. 2019 p 23. Tradução nossa).[13]

Neste sentido, os autores questionam, sobre o efeito implícito da Lei de Conway: "Quanta consciência o departamento de RH tem sobre sistemas de software? Os líderes do departamento de grupo que decidem como alocar o orçamento entre as equipes sabem do provável efeito de suas escolhas na viabilidade da arquitetura de software?"(SKELTON; PAIS. 2019 p 23. Tradução nossa)[14].

Portanto, existe a importância de envolver pessoas técnicas nas decisões sobre as responsabilidades do time e suas fronteiras, a forma do time, para isso, a figura do líder técnico é indispensável. Em suma, são dois lados da mesma moeda, a arquitetura organizacional e a arquitetura do software (SKELTON; PAIS, 2019).

Interessante notar que existe um entendimento amplo das pessoas envolvidas nos times de desenvolvimento além dos desenvolvedores, QAs, DBAs, OPs, POs, em outras abordagens. Notamos que estas se diferenciam ao não considerar a Lei de Conway.

Estas abordagens focam os demais inseridos atraídos pela perspectiva das regras de negócio (stakeholders) que compõem o software e menos a discussão sobre o impacto na arquitetura do sistema. Segundo Flores (2013) citando Evans (2004):

> …as equipes de desenvolvimento em debate não são compostas apenas por projetistas, engenheiros e desenvolvedores, devendo nas mesmas também serem inseridas todas as pessoas que, de alguma maneira, sejam afetadas pelo produto (FLORES 2013 *apud* Evans 2004).

Conforme o quadro abaixo, elaborado por Flores (2013), temos a descrição e exemplo de cada um dos atores presentes na citação acima.

[13] "if we have managers deciding … which services will be built, by which teams, we implicitly have managers deciding on the system architecture".

[14] How much awareness does the HR department have about software systems? Does the group department leaders deciding how to allocate budget across teams know of the likely effect of their choices on the viability of the software architecture? (SKELTON; PAIS. 2019 p 23).

Figura 6. Participantes das equipes de desenvolvimento

CATEGORIAS DE INTEGRANTES DAS EQUIPES DE DESENVOLVIMENTO DE *SOFTWARE*		
CATEGORIA	**DESCRIÇÃO GERAL**	**EXEMPLOS DE INTEGRANTES**
CLIENTES	Indivíduos que compram, utilizam ou são, direta ou indiretamente, afetados pelo *software*	• Clientes • Usuários do *Software*
GERENTES/LÍDERES	Pessoas eventualmente encarregadas do planejamento e do gerenciamento do processo de desenvolvimento de *software*	• Gerentes de Projeto • Líderes de Projeto • Líderes de Equipe
CONSTRUTORES	Indivíduos a quem é conferido o dever de produção do *software*	• Engenheiros • Desenvolvedores • Programadores
MENSURADORES DE QUALIDADE	Integrantes voltados à análise da qualidade do produto	• Avaliadores de Qualidade
FORNECEDORES DE INFRAESTRUTURA E SUPORTE	Participantes com propósito de obter e assegurar condições para o desenvolvimento do *software*	• Produtores • Financiadores • Promotores

Ativar o Win

FONTE: EQUIPES DE DESENVOLVIMENTO DE SOFTWARE: concepção, caracteres, espécies e fatores influentes no desempenho - 2013.

Publicado em 2013, o trabalho de Flores, “EQUIPES DE DESENVOLVIMENTO DE SOFTWARE: Concepção, Caracteres, Espécies e Fatores Influentes no Desempenho", foi o resultado da análise temática de trinta e sete artigos sobre engenharia de software.

Suas conclusões apontam para alguns fatores que reforçam a contribuição de Skelton e Pais (2019), no que diz respeito sobre a Lei de Conway como uma chave importante na organização de times. Vejamos abaixo:

> A construção de equipes de desenvolvimento de software constitui, atualmente, tarefa não sistematizada, sendo o seu êxito fortemente dependente das habilidades dos envolvidos em sua condução e dos próprios fatores impactantes no desempenho das equipes (FLORES, p.123, 2013).

Complementando a citação acima, a autora reforça a defasagem conceitual na construção de equipes:

> (...) não se evidencia a conscientização necessária a respeito dos diversos pontos que se deve considerar na construção das equipes em discussão (finalidade do desenvolvimento, peculiaridades do software a ser produzido, cultura e ambiente organizacionais, recursos materiais e imateriais disponíveis ao processo de desenvolvimento, heterogeneidade e balanceamento na composição, estruturação, estabelecimento de redundâncias, aspectos individuais particulares, metodologia de desenvolvimento, sistemática de trabalho, etc) (FLORES, p.124, 2013).

Em outro ponto, a autora conclui a ausência de critérios conceituais e teóricos que deveriam orientar a criação de equipes e acabam sendo substituídos por um senso de experiência pessoal de gestores e líderes:

> Em significativa parcela dos estudos analisados, não se observa a utilização de um protocolo para a construção de equipes; nos contextos destes estudos, equipes são construídas a partir das percepções pessoais de gerentes/líderes, considerados os recursos disponíveis e as peculiaridades dos projetos de desenvolvimento (FLORES, p.127, 2013).

Neste ponto, é notório que para a formação de times de desenvolvimento, deve-se pensar, de antemão, em critérios que consideram a correlação entre estrutura organizacional e arquitetura de software desejada. Falamos sobre tópicos gerais que devem ser considerados. Porém, vamos nos aprofundar mais sobre as contribuições de Skelton e Pais (2019), para entender os aspectos mais específicos.

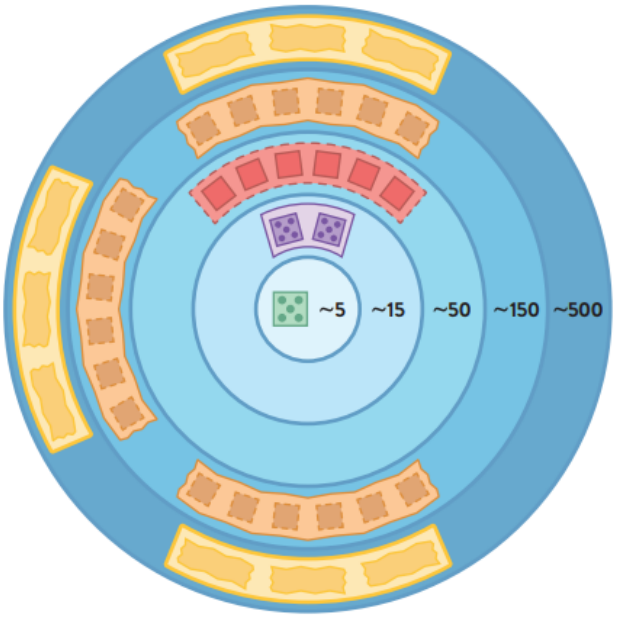
Os pontos que devem ser considerados são: tamanho do time, ciclo de vida, relacionamento entre o time e sua cognição. Por *times*, os autores partem do princípio que para melhor performance, eles devem ser pequenos, estimado entre 5 a 9 pessoas orientadas a compartilharem os objetivos como uma unidade (SKELTON; PAIS, 2019).

Outro motivo para as equipes serem reduzidas, está fundamentado na teoria *Número de Dunbar*[15]. Segundo esta teoria, as relações que possuem maior grau de confiança e conhecimento sobre cada pessoa, são compostas por até 5 indivíduos. Maior que este número, as relações vão ficando mais distantes, com um nível de conhecimento e confiança entre as pessoas menor

Vejamos a figura abaixo, o modelo “cebola” que ilustra o número de Dunbar:

[15] Segundo o Wikipédia (2022), “O **número de Dunbar** define o limite cognitivo teórico do número de pessoas com as quais um indivíduo pode manter relações sociais estáveis, ou seja, uma relação onde o indivíduo conhece cada membro do grupo e sabe identificar em que relação cada indivíduo se encontra com os outros indivíduos do grupo”. Disponível em: https://pt.wikipedia.org/wiki/numero_de_Dunbar> Acessado em: 05/22.

Figura 7 . Número de Dunbar



Fonte: Team Topologies: Organizing Business and Technology Teams for Fast Flow - 2019[16]

Para tratar sobre a estrutura de times nas organizações, a abordagem utilizada pelos autores é chamada de *Team-First.* Nesta, o time é concebido como a unidade mínima da organização responsável pelas operações e entregas.

Portanto, nada deve ser atribuído individualmente, sempre devemos pensar orientado pela noção de time. O número de Dunbar, segundo os autores, orienta esta abordagem, uma vez que mudanças realizadas na arquitetura de um software, devem levar em consideração o limite das interações entre pessoas (SKELTON; PAIS, 2019).

Um aspecto comportamental importante, é que uma vez que o time está definido e unido, ele precisa desenvolver o que os autores chamam de *continuity of care,* ou seja, manter o frequente cuidado com o software. Para isso, é necessário que o time se assuma como proprietário do software, no sentido do time ser responsável pelo código por assumir seu cuidado, não dono no sentido de excluir os demais. Os autores ressaltam: “Pense no código como jardinagem, não como policiamento”[17](SKELTON; PAIS, 2019, p. 37, tradução nossa).

Outro aspecto comportamental, é desenvolver o mindset *Teams-First*, que significa colocar as necessidades acima das necessidades individuais. Para os autores, pessoas que são

[16] *Ibidem, p. 34*
[17] Think of code as gardening not policing (SKELTON; PAIS, 2019, p 37).

inadequadas para trabalhar em grupo ou relutantes em colocar os interesses do time abaixo dos interesses pessoais, devem ser afastadas antes que afetem o trabalho do time ou tornem o time tóxico.

Percebemos que a abordagem *Team-First* deve ser vivenciada não só por pessoas com capacidade de decisão, como gestores e tech leads. Mas por cada membro que compõe o time.

Quais são os pontos elencados pelos autores e que contribuem para formar o mindset de *Team-First*?

> Chegar em tempo para as reuniões; manter discussões e investigações; encorajar o time a focar nos seus objetivos; ajudar a desbloquear outros membros antes de iniciar um novo trabalho; mentorar novas ou pessoas com menos experiência e, por último, ao invés de querer ganhar nos argumentos, concordar em explorar as opções (SKELTON; PAIS, 2019, p. 37, tradução nossa).

Neste ponto, já temos a justificativa dos ganhos de se ter times pequenos, unidos e conscientes da abordagem *team-first.* Agora, é necessário descrever o que os autores chamam de carga cognitiva. No contexto organizacional, é a dimensão do sistema que se espera que um time possa lidar. No sentido psicológico, é a quantidade total de esforço mental usado pela memória no trabalho. Frente a isto, para o bom desempenho do time, a carga mental não deve ser alta. (SKELTON; PAIS, 2019).

A carga cognitiva deve ser distribuída para que não sobrecarregue os times. E contratar mais pessoas esperando que isto diminuirá esta carga, dado o esforço comunicativo do desenvolvimento, não é a solução e resultará no problema tratado pela lei de Brooks que mencionamos no início que diz: “a adição de mais homens, portanto, aumenta, e não diminui o cronograma.”.

Para trabalhar o conceito de carga cognitiva, os autores evocaram John Sweller(1988), este define três tipos de carga cognitiva que estão presentes no ambiente de desenvolvimento. Vejamos abaixo:

Tabela 2 - Carga cognitiva

Tipo de Carga Cognitiva	Descrição	Como amenizá-la
Carga cognitiva intrínseca	Relacionada aos aspectos fundamentais para o espaço do problema. Ex: “O que é a estrutura de uma classe em	Através de treinamento, escolha de boas tecnologias, pareamento entre programadores,

	Java", "Como eu crio um novo método"	
Carga cognitiva estranha	Relacionada ao ambiente em que a tarefa começa a ser feita. Ex: "Como faço o deploy deste componente?", "Como eu configuro este serviço?"	Eliminar tarefas chatas ou supérfluas que agregam pouco valor, mas consomem muita memória e que podem ser frequentemente automatizadas.
Carga cognitiva pertinente	Relacionada aos aspectos de tarefas que precisam de atenção especial para aprender ou à alta performance.	Com a redução das duas acima, o time pode ficar com mais espaço para a cognitiva pertinente, a qual se concentra em tarefas que agregam valor.

Fonte: **Team Topologies**[18]

Segundo Skelton e Pais (2019), não há uma fórmula correta para captar o nível exato da carga cognitiva exigida, porém, segundo os autores, podemos definir domínios de complexidade fundamentado em quão complexo é um problema que o time está tentando resolver.

Neste sentido, os autores sugerem limitar a quantidade e o tipo de domínio, que pode impactar positivamente na definição das fronteiras e responsabilidades dos times. Um time, por exemplo, não poderia ficar sobrecarregado com dois domínios complexos, que faria aumentar sua carga cognitiva (SKELTON; PAIS, 2019).

Com domínios complicados ou complexos devidamente distribuídos, a motivação do time aumenta, bem como a velocidade das entregas e a solução de problemas. Com domínios complicados ou complexos mal distribuídos, o time trabalha com pouca motivação, baixa colaboração, trocas constantes e desengajamento de indivíduos (SKELTON; PAIS, 2019).

> Ao invés de projetar um sistema abstrato, nós precisamos projetar os sistemas e os contornos do software para ajustar na carga cognitiva disponível dos times de entrega.
> (...) Ao invés de escolher entre uma arquitetura monolítica ou uma arquitetura de microsserviços, projetar o software para se encaixar ao máximo na carga cognitiva da equipe. Somente então conseguiremos ser sustentáveis, seguros, rápidos na entrega de software.

[18] SKELTON, Matthew; PAIS, Manuel. **Team Topologies**: organizing business and technology teams for fast flow. Portland: It Revolution, 2019. 216 p.

5.4 Quatro tipos de topologias

Os quatro tipos definidos como topologias fundamentais, servem para sistematizar e ordenar as interações entre as equipes dentro das organizações. A intenção é enxugar e simplificar nestes quatro tipos as variedade de equipes de desenvolvimento.

Para os autores, se cuidadosamente utilizadas, a combinação destas topologias é o suficiente para construir softwares com qualidade (SKELTON; PAIS, 2022)

Vejamos quais são os quatro tipos:

- **Stream-aligned team**
- **Enabling Team**
- **Complicated-subsystem team**
- **Platform team**

Antes de seguir com mais detalhes sobre cada tipo de equipe, em uma tradução livre, podemos definir a lista acima respectivamente como:

- Equipe Fluxo-alinhado;
- Equipe Permitindo;
- Equipe de subsistemas complicados;
- Equipe de Plataforma.

Isto posto, seguimos utilizando o nome definido no idioma original.

O *stream-aligned team* é uma equipe alinhada para valorizar o fluxo de trabalho, no cenário em que o fluxo é um contínuo e que está alinhado com o domínio de negócio e a capacidade organizacional. Não importa a dimensão, o produto, a funcionalidade e a quantidade de clientes, esta equipe é empoderada para entregar valor ao cliente com velocidade, segurança e quando possível é independente de outras mãos para realizar partes de seu trabalho (SKELTON; PAIS, 2019).

Esta equipe, no modelo dos autores, é a equipe primária dentro da organização, sendo as demais, como enabling team e platform team, suportes para que o fardo do *stream-aligned* team seja menos pesado. Por exemplo: o *enabling team* auxiliar na aquisição de capacidades faltantes, na aplicação das melhores práticas. O *platform team*, atuará para auxiliar o stream-aligned team a reduzir a carga cognitiva das atividades de desenvolvimento através do provisionamento de serviços fáceis de serem consumidos e que serão disponibilizados.

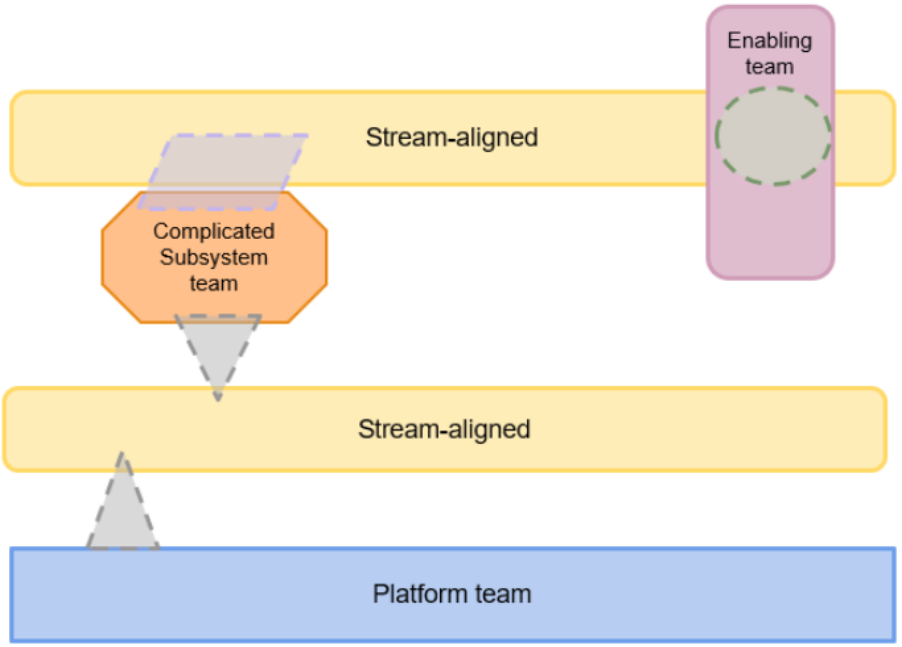
Outra característica do *stream-aligned team* é sua proximidade com o cliente, por isso ele deve ter acesso ao monitoramento do sistema em produção para que possa receber feedbacks rapidamente e solucionar os problemas de modo efetivo.

É importante salientar que esse tipo de equipe pode coexistir dentro de uma organização e pode tomar a forma de uma espécie de micro empresa com foco e propósitos bem definidos (SKELTON; PAIS, 2019).

Quais são as capacidades de um *stream-aligned team*? São listadas as seguinte:

- Segurança da aplicação;
- Análise de viabilidade comercial e operacional;
- Design e arquitetura;
- Desenvolvimento e código;
- Infraestrutura e operabilidade;
- Métricas e monitoramento;
- Gerenciamento de produto;
- Teste e qualidade;
- Experiência do usuário.

FIGURA 8 - Quatro tipo de topologias e como interagem



Fonte: Team shape templates[19]

[19] Os templates para modelos de topologias estão disponibilizados em: https://github.com/TeamTopologies/oS Team-Shape-Templates

Observação feita por Skelton e Pais (2019), é sobre a importância das competências acima não serem concentradas como papéis individuais. O ideal é uma mistura entre poucos especialistas que poderiam liderar e uma quantidade de pessoas generalistas. O risco de ter apenas especialistas, é de formar um gargalo porque o trabalho de um depende de outro especialista que estaria ocupado.

Por fim, os autores reforçam que a escolha pelo nome *stream-aligned team* ao invés de equipe de produto ou de equipe de feature, é que *stream* reforça o senso de fluxo. O que se espera do comportamento deste tipo de topologia? Vejamos alguns listados pelos autores[20]:

- Ter um fluxo contínuo de entregas de features[21];
- Ter capacidade para atuar rapidamente em correções baseado no feedback das últimas mudanças;
- Utilizar abordagem experimental para evoluir o produto esperando constantemente aprender e adaptar;
- Reduzir o máximo possível a atuação de outros times dentro do seu fluxo de trabalho;
- Sentir que tem um objetivo ou que está no caminho para alcançar, estes objetivos são: autonomia, domínio e propósito.

Os *enabling teams*, são compostos por especialistas em um dado domínio técnico ou domínio de produto. Se trata de uma equipe transversal que auxilia sugerindo as melhores práticas, ferramentas, frameworks, entre outros tipos de colaboração.

Skelton e Pais (2019) citam como exemplo, o stream-aligned team pelo seu fluxo de entrega com proximidade com o cliente, não tem muitas vezes espaço para pesquisar, aprender e praticar novas habilidades. Neste sentido, que estará de prontidão para responder dúvidas, sugerir a melhor ferramenta de acordo com o sistema, será a *enabling teams*. Na Figura 8 acima, podemos visualizar este exemplo que, em suma, trata-se de transferência do conhecimento.

---

[20] SKELTON, Matthew; PAIS, Manuel. **Team Topologies**: organizing business and technology teams for fast flow. Portland: It Revolution, 2019. 216 p.

[21] Feature, em tradução livre, significa característica. Em desenvolvimento de software, feature é um requisito que será construído e integrado no software. Pode ter ser uma melhoria de algo que já exista ou uma solicitação de algum cliente.

São equipes de natureza colaborativa e podemos defini-los como consultores técnicos para clientes internos e externos, são lideranças montadas para servir outras equipes. (SKELTON; PAIS, 2019). Esta equipe deve colaborar para que as demais possam criar autonomia e aumentar a independência para a resolução de problemas e por isso, ela não deve ser uma dependência permanente, mas curta e efetiva.

Vejamos o comportamento esperado para estas equipes:

- Entender as necessidades das equipes e agregar com mais colaboração quando necessário;
- Trazer boas e más notícias em relação às escolhas técnicas;
- Concentrar em auxiliar outras equipes a criar uma melhor dinâmica;
- Promover aprendizagem, atuar como curadores que facilitam a apropriação e compartilhamento de conhecimento dentro da organização.[22]

Quanto ao *Complicated-subsystem teams,* segundo os autores Skelton e Pais (2019), é um time não permanente, vejamos :

> Um *complicated-subsystem é* responsável pela construção e manutenção de uma parte do sistema que depende fortemente do conhecimento de especialistas (...) O objetivo deste time é reduzir a carga cognitiva do stream-aligned teams trabalhando no sistemas que inclui ou utiliza o *complicated-subsystem* (SKELTON; PAIS. 2019 p 91. Tradução nossa).

Este tipo de equipe é criada quando um subsistema necessita de ajuda especializada, podemos visualizar esta interação na figura 8. Vejamos o tipo de comportamento esperado[23]:

- Alta colaboração com stream-aligned durante as fases de desenvolvimento. Reduzir as interações e focar no sistema e na sua evolução depois, quando estiver estável;
- Entregar velocidade e qualidade para o subsistema quando for iniciado o desenvolvimento pelo stream-aligned teams;
- Priorizar corretamente e entregar o trabalho respeitando as necessidades do stream-aligned teams que usa o complicated subsystem.

---

[22] *Ibidem, p. 88*
[23] *Ibidem, p. 91*

Por fim, vejamos o tipo *Platform teams*. O seu objetivo é dar o suporte necessário através de serviços que permitam que o stream-aligned teams trabalhe com uma substancial autonomia. A equipe platform providencia serviços internos para reduzir a carga cognitiva que poderia ser uma necessidade do stream-aligned teams (SKELTON; PAIS, 2019).

Vejamos a definição de platform:

> Uma plataforma digital é uma fundação de APIs self-service, ferramentas, serviços, conhecimento e apoio que são organizados como um produto interno atraente. Equipes autônomas de entrega podem usar a plataforma para entregar produtos em um ritmo mais rápido, com coordenação reduzida (SKELTON; PAIS. 2019 p 92. Tradução nossa).

A maneira como as equipes de platform disponibilizam seus serviços, deve corresponder com a frase “ease of use”, que podemos traduzir como “facilidade de usar”, isso reflete como os clientes internos e externos devem sentir quando consomem produtos desenvolvidos pelo platform. Na figura 8, podemos ver como o time interage. Abaixo, veremos o comportamento esperado[24].

- Forte senso de colaboração com o stream-aligned teams para entender suas necessidades.
- Confia na velocidade na prototipagem rápida e envolve o stream-aligned teams para a rapidez do feedback sobre o que é para ser trabalhado e o que não é.
- Forte foco na usabilidade e confiabilidade para seu serviços e regularmente avalia se seus serviços ainda estão adequados com o propósito e utilizável.
- Entende que a adoção interna de novos serviços, como novas tecnologias, não é imediata, mas envolve uma longa curva de adoção.

Para exemplificarmos, o platform é uma entidade que providencia serviços que podem simplesmente ser consumidos via API: provisionamento de máquinas ou containers, configuração de redes, monitoramento. Em empresas de grande porte, equipes de plataforma podem colaborar com outras equipes do mesmo tipo. Em suma, o foco deve ser, além da experiência do usuário, mas também o que foi chamado pelos autores de DevEx, que significa developer experience ou experiência do usuário. (SKELTON; PAIS, 2019).

Segundo os autores, se quisermos entregar software com qualidade e velocidade, devemos evitar ilhas ou silos em que cada pessoa executa uma função. O esperado no contexto de equipe de plataforma é que seja multidisciplinar e transversal entre as equipes

[24] *Ibidem, p. 94*

streams-aligned que são compostas de pessoas com diversas habilidades. Portanto, é contraproducente uma equipe de plataforma separada em que cada membro possui uma função individual.

Por fim, a clareza dos modelos de tipologias e os propósitos contidos em cada uma delas, quando identificados, trazem ganhos para a organização, como salientado abaixo:

> Algumas organizações poderiam se beneficiar aumentando a clareza acerca da definição e propósito de suas equipes. Na verdade, nós pensamos que muitas organizações veriam maiores ganhos efetivos mapeando suas equipes para cada um dos quatro tipos fundamentais de topologias; isto é, identificar qual das quatro fundamental topologias representaria a melhor maneira de trabalhar de cada equipe e então mudar a missão para adotar o propósito e o padrão de comportamento dessa topologia (SKELTON; PAIS. 2019 p 104. Tradução nossa).

A adoção e identificação dos quatro tipos de topologias, pode ser adequada tanto às equipes centralizadas em um espaço físico quanto no contexto do trabalho remoto.

5.5 Organização de times de desenvolvimento no contexto *remote-first*

Entre muitos aspectos que transformaram a vida social a partir da pandemia de COVID-19, o modelo de trabalho no mercado de tecnologia teve uma transformação que ficou consolidada. Este modelo é o trabalho remoto. Anteriormente existia, porém não fazia parte da normalidade na tecnologia. Era aplicado em situações específicas, como alguma exceção por motivo pessoal em que a pessoa não pôde ir ao escritório, outras como bônus de ser remoto algumas vezes por semana, porém nunca tinha sido consolidado.

Com sua consolidação a partir de 2019, passou a ser também intitulado como *remote first.* Neste modelo, todas ou a maioria das pessoas não trabalham em um escritório centralizado, mas sim distribuídas geograficamente e conectadas pela internet.

No entanto, o trabalho remoto impõe algumas variáveis se partirmos do pressuposto que sistemas são reflexo da maneira em que as equipes se organizam e se inter-relacionam. Neste sentido, é importante racionalizar que da mesma forma que equipes não devem ser organizadas apenas por feelings e para suprimir demandas, o modelo remoto também precisa ser sistematizado e racionalizado para não ser uma extensão do trabalho presencial.

Uma boa leitura dos modelos de trabalho remoto foi feita em 2020 por Matt Mullenweg, criador do Wordpress. Ele define cinco níveis do trabalho remoto[25].

---

[25]Mais detalhes: Distributed Work's Five Levels of Autonomy. Disponível em: https://ma.tt/2020/04/five-levels-of-autonomy/.

O primeiro nível, no qual o autor identifica que é a situação da maioria das empresas (considerando que o artigo escrito no início de 2020), é que não existe uma relação amigável com o trabalho remoto. São empresas, que no início da pandemia, foram pegas desprevenidas e tiveram que, às pressas, alterar o modelo de trabalho.

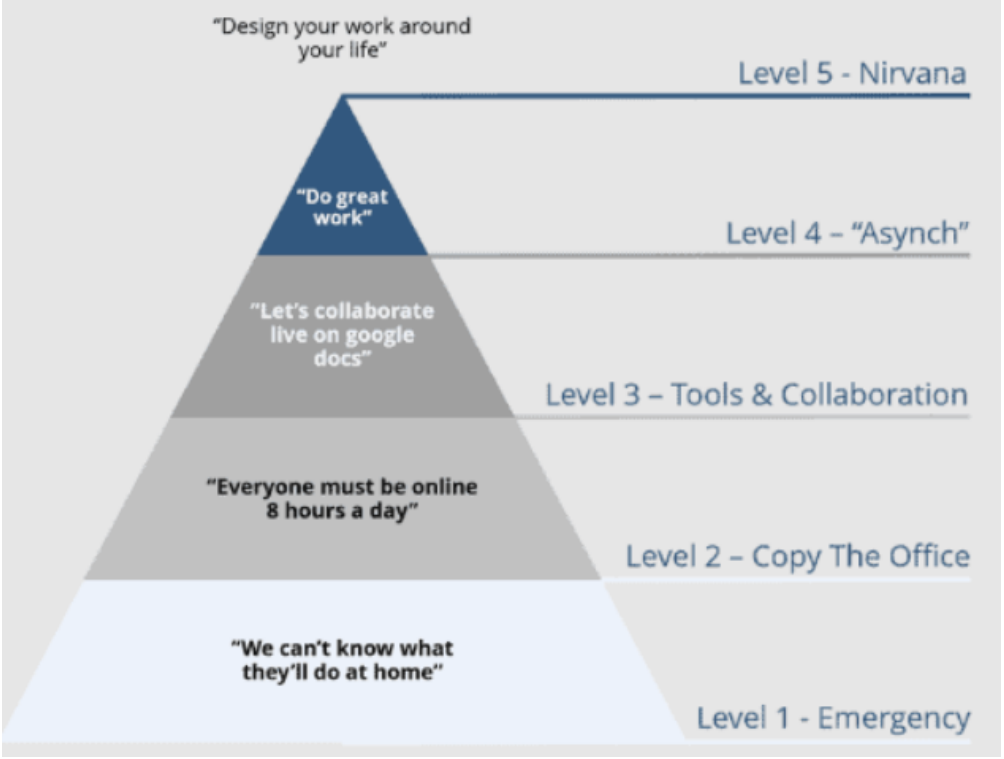
O segundo nível, é o da recriação do ambiente do escritório dentro do trabalho remoto após aceitá-lo por conta da pandemia. Neste nível, existe uma preocupação em demasia com a produtividade por parte da gerência. A comunicação esperada nos chats, emula a comunicação da vida no escritório, as interrupções são constantes. Neste nível, estão as empresas que instalam softwares espião para monitorar a produtividade dos trabalhadores.

No terceiro nível, para o autor, é quando começa a aproveitar as benesses do trabalho remoto. As empresas começam a investir em aparelhos melhores, a comunicação escrita é levada para um nível maior de atenção a qual permite que as pessoas compartilhem e elaborem documentos juntos durante uma reunião.

No quarto nível, as relações e comunicação são assíncronas, para o autor, neste nível o trabalho é avaliado pelo que foi produzido e não como e em quanto tempo. Neste ponto, estabelece uma relação de confiança entre a equipe que gera engajamento, as reuniões respeitam horário pré estabelecido, a retenção de funcionários é maior e existe maior autonomia para realizar o trabalho.

O último nível, é o ideal e o autor diz não ser totalmente alcançável. Neste, o desempenho é melhor que qualquer emprego presencial poderia oferecer. O trabalho é realizado com eficácia e sem esforço, existe preocupação da empresa com a saúde mental e bem estar dos trabalhadores, existe criatividade, diversão e as pessoas dão o melhor de si.

Figura 9 - Níveis do trabalho remoto



Fonte:Matt Mullenweg's & Automattic's Five Levels Of Remote Work. Boundless by Paul Millerd

Tratando da importância da comunicação entre o time, da interação, da abordagem team first, o nível três permite inter relações importantes para tipologia em times remotos.

Do ponto de vista da Lei de Conway, como esta nova estrutura de comunicação e relação pode impactar na estrutura do sistema? Como podem ser pensadas as topologias que vimos até então e que se baseiam na Lei de Conway?

Para responder essas questões, em 2022 Matthew Skelton e Manuel Pais lançaram *Remote Team Interactions Workbook.* A intenção é pensar como analisar e aplicar a topologia adequada para times remotos.

Os autores iniciam dizendo sobre um equívoco que muitas vezes acompanha o modelo remoto, que é acreditar que apenas utilizar ferramentas de comunicação como chats ou de videochamadas, são suficientes para manter o time engajado e garantir boas entregas. Para não cometer esses equívocos, os autores citam cinco chaves utilizadas que fundamentam os times remotos na Google. Vejamos:

1. **Segurança psicológica:** As equipes podem enfrentar riscos sem sentimento de insegurança?

2. **Confiabilidade:** As equipes podem contar uns com os outros?
3. **Estrutura e clareza:** Os objetivos, regras e plano de execução estão claros?
4. **Significado de trabalho:** O trabalho é pessoalmente importante para a equipe?
5. **Impacto do trabalho:** A equipe acredita que o trabalho importa?

Estas perguntas, auxiliam a criar uma definição de maneiras de se trabalhar, gerir expectativas, reconhecer padrões e modos de comportamento e interação entre as equipes. Ajudam também a reduzir a carga cognitiva presente na equipe (SKELTON; PAIS, 2022).

Os autores retomaram o conceito de *carga cognitiva*, mas desta vez com algumas perguntas para medir e entender o contexto da equipe no trabalho remoto e reduzir tal carga. Eles estimam para cada pergunta a classificação de 1(fraco) a 5 (muito bom). As perguntas cobrem os seguintes temas: building, testes, deploy, suporte, compliance e segurança. Vejamos as perguntas dos autores (SKELTON; PAIS, 2022).

1. Como é a experiência de construir seus serviços?
2. Como é a experiência de testar seus serviços?
3. Como é a experiência de fazer o deploy dos serviços?
4. Como é a experiência de operar seus serviços?
5. Como é a sua experiência em estar de plantão para seus serviços?
6. Como é a experiência em lidar com as regulamentações e compliance?
7. Gostaria de comentar como é sua experiência de modo geral ?

Para comunicar as responsabilidades e o foco da equipe, a disponibilidade e comunicação entre equipes, os autores sugerem para equipes remotas, que utilizem a abordagem *Team API* como ponto de partida. É uma analogia entre o que se espera de uma boa API[26] que segue as boas práticas do desenvolvimento de software e o que se espera da comunicação e interação entre equipes. Vejamos na tabela abaixo a associação entre a abordagem de uma API e a extensão para equipes.

[26] API é sigla para: Application Programming Interfaces. Em português é: Interfaces de Programação de Aplicações. Para uma ideia simplificada, podemos dizer que são programas que ficam entre o que vemos na tela e um servidor que guarda as informações, as APIs ficam nesta interface, levando para um banco de dados as ações do usuário, como: solicitar, excluir, deletar, incluir e atualizar informações. Cada uma dessas solicitações, são chamadas de requisições e o local que elas devem ir são identificadas como endpoint. As APIs são como mensageiras que buscam em um servidor e retornam essas ações de acordo com nossas solicitações.

Tabela 3 - Team API

API	Equipe
Códigos e artefatos da equipe	endpoints, bibliotecas, interfaces do usuário, clients produzidas pela equipe.
Versionamento	Como o time comunica as mudanças de seus códigos e serviços.
Wiki	Guias de como lidar com o software que é responsabilidade da equipe.
Práticas e princípios	Os modos que a equipe prefere trabalhar
Comunicação	As ferramentas utilizadas para estabelecer comunicação remota.
Road maps e prioridades	Informar no que a equipe está trabalhando no momento, definir o nível de prioridades das tarefas.
Outros	Qualquer coisa que seja utilizada para estabelecer comunicação entre uma equipe e outra.

Fonte do autor

Tendo esta abordagem como ponto de partida, segundo os autores, ajuda a melhorar a clareza de propósitos e ajudar outros grupos a entender como as equipes se encaixam dentro da organização (SKELTON; PAIS, 2022).

No contexto *remot-first*, os chats são ferramentas aliadas para comunicar status de entregas, neles se promove a interação humana na organização, porém as mensagens devem ser muito bem contextualizadas, com insumos para o bom entendimento que colaboram para a redução da carga cognitiva.

As palavras devem ser objetivas, as frases sem brechas para suposições, sempre que possível reportar *em que* está trabalhando, *por que* você está trabalhando nisto, *como* seu trabalho está sendo concluído e *quando* estará completo (SKELTON; PAIS, 2022).

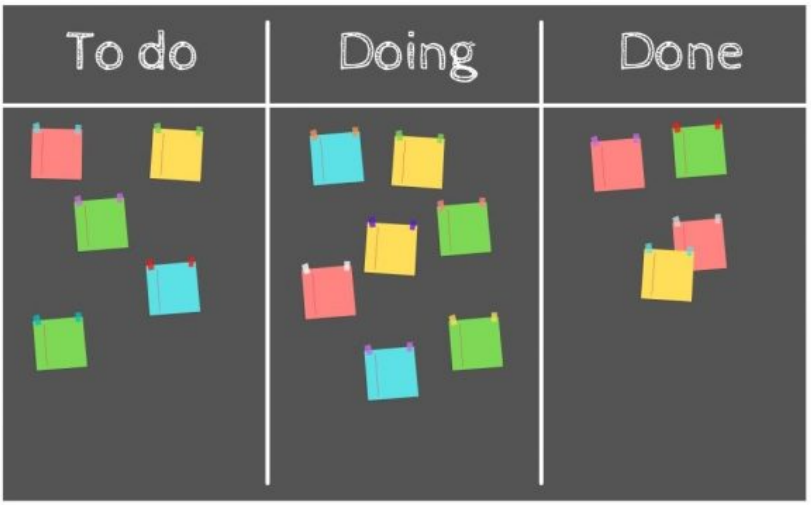
5.6 Identificar e reduzir as dependências (desacoplar as equipes)

Identificar e reduzir as dependências entre times, separá-las entre o que é ou não bloqueante. As bloqueantes impedem o fluxo de trabalho e estas devem ser rastreadas e mitigadas. Dar visibilidade para as dependências e o que cada time trabalha, segundo os autores, facilita a fluidez do processo de desenvolvimento. Essa visibilidade pode ser feita através de um Kanban que também está disponível online.

Vejamos brevemente o que é o Kanban:

> Do japonês, Kanban é literalmente traduzido como *quadro de sinal* ou *sinal visual*. O quadro Kanban mais simples possui três colunas – “Pedido”, “Em Progresso” e “Concluído”. Quando construído e gerenciado adequadamente, ele serve como uma central de informações em tempo real, destacando os gargalos no sistema e qualquer outra coisa que possa atrapalhar as práticas de trabalho (KANBANIZE, 2022).

Figura 10 - QUADRO KANBAN



Fonte: https://kanbanize.com/pt/recursos-kanban/primeiros-passos/o-que-e-kanban. Cada coluna está escrito: “Fazer”, “Fazendo” e “Feito”

Segundo os autores, a análise de características dos times ao invés de ser top-down, ela deve passar por cada time e espalhar as informações sobre cada um para todas as direções tornando as APIs acessíveis e fáceis de serem consumidas. Os autores reforçam que quanto mais evidentes e acessíveis as informações, menor a carga cognitiva.

Os teams APIs podem deliberar sobre como eles preferem ser vistos por outros times e como desejam se comunicar com os demais times. Podem definir sua própria API de maneira independente umas das outras. Para os autores, somado a facilidade de acesso, leva a aumentar a clareza da comunicação e interação entre outros times para sanar dúvidas e colaborar na resolução de problemas, tornando mais simples de serem consumidos por pessoas de fora da equipe (SKELTON; PAIS, 2022).

Interessante notar, que no modelo de trabalho remoto, o foco na entrega de sistemas de qualidade e eficientes, continuam sendo uma meta. A questão é como será o arranjo para tal. E neste caso, os autores sempre reforçam a questão da comunicação, interação e autonomia, esses aspectos estão contemplados dentro da abordagem Team API, que continuaremos vendo.

Para explicitar como a equipe atua, é sugerido criar uma espécie de documentação/template com as seguinte informações (SKELTON; PAIS, 2022)[27]:

Team API

- *Nome e foco da equipe*
- *Tipo da equipe.*
- *Nós providenciamos serviços para quais outras equipes?*
- *Software mantido pela equipe.*
- *Versões abordadas.*
- *Wiki e termos de busca.*
- *Canais do chat que estamos inseridos.*
- *Horário da daily.*
- *Nossos serviços e sistemas.*
- *Maneiras de trabalhar.*
- *Equipes com que nós interagimos atualmente (nome da equipe, modo de interação, propósito e duração).*
- *Equipes com que esperamos interagir em breve (nome da equipe, modo de interação, propósito e duração).*

Um dos pontos a serem considerados em remot-first, à comunicação, interação, bem estar da equipe e clareza e divulgação das informações, é a dimensão das equipes. Em *Remote Team Interactions Workbook: organizing business and technology teams for fast flow (2022)*, entre outros conceitos retomados, os autores trazem à tona o Número de Dunbar que falamos no capítulo anterior, o qual define o limite máximo de pessoas para que se estabeleça relações significativas. A partir de então, veremos como ele pode ser contextualizado na modalidade de trabalho remot-first.

Robin Dunbar, é um antropólogo contemporâneo e aplicando sua teoria nas redes sociais, como Facebook e Twitter, o resultado do limite de interação para um usuário típico é de 150 pessoas (SKELTON; PAIS, 2022). Em outras palavras, a construção de relações deixa de ser proporcional à quantidade de pessoas após identificar o crescimento que ultrapassou a fronteira das relações significativas. Neste sentido, à medida que o número de pessoas

[27] Os autores, SKELTON e PAIS(2022), disponibilizaram as informações em forma de um template que pode ser consultado e utilizado, acessando o seguinte link: https://github.com/TeamTopologies/Team-API-template.

aumenta dentro de uma organização, uma possibilidade para garantir as relações significativas, é fracionar.

Qualidades como confiança e coesão são resultantes de grupos em que as fronteiras de interação foram identificados e estabelecidas. Aumenta a eficiência da organização e evita que se estabeleça uma divisão que os autores chamam de “nós e eles” (SKELTON; PAIS, 2022). O contrário é uma realidade, grupos grandes podem ser prejudicados pela falta de coesão e confiança, bem como estabelecer separações que vão de encontro com a ideia de *team first*, impactando na velocidade e qualidade das entregas.

Como estamos lidando com trabalho remoto, se impõem a seguinte questão, como organizar e distribuir estes grupos nos canais de comunicação online? Para responder, podemos utilizar o conceito que Skelton e Pais (2022) chamaram de *espaços onlines*.

Para os autores, a chave é desenharmos estes espaços levando em consideração as fronteiras de confiança que foram identificadas. Cada espaço contém canais de comunicação, e grupos (estes possuem menos pessoas). A recomendação é que evitemos que sejam construídos espaços por funções, pois assim, afetaria a velocidade do fluxo de mudança. Por exemplo, um grupo e canal com apenas desenvolvedores, com a comunicação fechada entre eles, a equipe de qualidade não teria a quem reportar inconformidades e os desenvolvedores deixariam de avisar alguma atualização lançada que afetasse os ambientes de testes.

Neste sentido, existe dentro de espaços onlines, podemos utilizar os canais de modo que membros de diferentes grupos responsáveis por algum tipo de serviço, compartilhem informações, respondam dúvidas e relatem impedimentos. Considerando que na abordagem remote-first as ferramentas de interação são o meio de comunicação, é importante que a escolha do nome do canal seja clara e propositiva, de modo que facilite reconhecer as interações entre os times.

O estilo de comunicação no chat também deve ser claro, isso impacta na eficiência da organização. “Para uma organização ser efetiva, especialmente no contexto do trabalho remoto, é essencial para o propósito de qualquer canal de comunicação ser claro e óbvio.”(SKELTON; PAIS. 2022 p 46. Tradução nossa).

Para os autores, devemos privilegiar canais públicos, para que as pessoas peçam auxílio e compartilhem dúvidas nestes canais e não individualmente. Essa prática reforça a ideia de equipe e compartilha a responsabilidade sobre o produto entre a equipe.

Aliados à comunicação organizada, a colaboração também deve ser organizada e sistematizada, assim como no ambiente não remoto. Os autores tratam como mito que o

simples apelo para mais comunicação e mais organização é suficiente para assegurar o fluxo eficiente e que entrega sistemas com qualidade.

> “Mais colaboração” é frequentemente sugerida como uma solução para a ineficácia organizacional, mas, na realidade, colaboração precisa ser cuidadosamente tratada para evitar efeitos inesperados e sobrecarga cognitiva, (SKELTON; PAIS. 2022 p 37. Tradução nossa).

.

A maneira como é organizada a colaboração entre os times, passa por três abordagens identificadas por Skelton e Pais (2022) :

- **Colaboração**: dois times trabalham juntos por um período específico de tempo dentro de um alvo bem definido.
- **X-as-a-Service**: um time fornece algum tipo de serviço e outro time consome algo.
- **Facilitador**: um time ajudando outro time a corrigir ou descobrir coisas.

A intenção é explicitar o propósito da interação entre os times, para que a equipe atue de maneira ordenada e compreenda sua atuação no fluxo de desenvolvimento reduzindo a comunicação excessiva e desorganizada e a colaboração que pode prejudicar.

As três abordagens acima são destacadas por ajudar a identificar déficit técnico entre os times, corrigir falhas na comunicação e na inter-relação. A colaboração, por exemplo, é aplicada em ciclos temporários mais curtos. Por outro lado, se ela é recorrente e dura longos períodos, é um sinal que devemos levar em consideração e inferir que deve haver débito técnico, sobrecarga de atividades, falha na especificação, entre outros motivos que podem emergir no contexto organizacional.

O modo de interação, facilitador, são times que atuam juntos de modo similar a colaboração, porém com objetivo pontual e por período curto também. Caso contrário, é ineficiente porque se for de interação recorrente e frequente, é inefetivo e torna-se necessário identificar o que pode estar alterando o propósito.

O modo de interação mais longo entre times, é o x-as-a-service. Podemos inferir que esta notação que os autores criaram para definir esse modo, é inspirada na maneira que os serviços de nuvem são chamados. Exemplo. PaaS - Platform as a Service; SaaS - Software as a Service; IaaS - Infrastructure as a Service. X as a Service, o time que provisiona determinado serviço tem uma com time que consome, e o provedor deve estar aberto para

feedbacks e ambos devem cuidar para que o limites entre cada time estejam no lugar correto (SKELTON; PAIS, 2022).

De modo geral, Skelton e Pais (2022), afirmam que todos os conceitos trabalhados até então, como limites e fronteiras entre times, carga cognitiva, reduzir dependências, entre outros, concentra-se no fluxo de desenvolvimento com a seguinte intenção:

> (...) ajudar a tornar o trabalho assíncrono e remoto mais eficaz precisamente porque as pessoas em locais separados têm o contexto, a confiança e o foco para serem capazes de trabalhar de forma eficaz: em parte de forma independente e em parte sincronizando as principais decisões e explorações (SKELTON; PAIS. 2022 p 44. Tradução nossa).

Por fim, a comunicação e interação tem sido o core na engenharia de software que se dedica ao tema de equipes de desenvolvimento e, ao longo do presente trabalho, desde Brooks, passando Conway e reiterada nos trabalhos de Skelton e Pais (2022), no contexto do remot-first este core é ressignificado dentro dos espaços onlines.

6. PROCEDIMENTOS METODOLÓGICOS

O presente capítulo busca apresentar a caracterização da pesquisa, elucidando sua natureza e o procedimento metodológico adotado, bem como a estimativa para sua conclusão. Por fim, tratamos da delimitação que apresenta o escopo da pesquisa, reiterando o objetivo final da sua entrega e as etapas para a conclusão.

6.1 CARACTERIZAÇÃO DA PESQUISA

Trata-se de uma pesquisa de natureza teórica, isto é, que visa a articulação de conceitos e a sistematização de ideias, através da abordagem qualitativa (ALMEIDA, 2014). A pesquisa será realizada sob uma perspectiva exploratória, a fim de contribuir com o aumento do conhecimento sobre o tema. Para tal, o procedimento adotado será de pesquisa bibliográfica.

Para melhor esclarecimento sobre o procedimento bibliográfico adotado, devemos considerar: “este tipo de estudo toma como objeto apenas livros e artigos científicos, tendo normalmente a finalidade de buscar relações entre conceitos, características e ideias, às vezes unindo dois ou mais temas” (ALMEIDA, 2014, p.33).

Ainda sobre o procedimento bibliográfico:

> A pesquisa bibliográfica constitui o ato de ler, selecionar, fichar, organizar e arquivar tópicos de interesse para a pesquisa em pauta. É a base para as demais pesquisas sendo uma síntese referente ao tema abordado. Consiste em apresentar e comentar o que outros autores escreveram sobre o tema, enfatizando as diferenças ou semelhanças que existem entre os conceitos (METODOLOGIA-MAXWELL, 2005. p 3 *apud* Rovery, 2000).

Para a execução desta pesquisa, tivemos como fontes documentos, tais como leitura de entrevistas, ensaios, juntamente ao referencial teórico, o qual envolve livros, artigos e outros trabalhos bibliográficos dentro da tecnologia da informação especificamente sobre a composição de times de desenvolvimento.

O tempo estimado para a construção da pesquisa foi de seis meses. A estimativa é baseada no decorrer de 1 semestre como parte da disciplina Projeto de Aplicação de GTI III, que será ofertada no segundo semestre do ano de 2022. A entrega do resultado da pesquisa foi no final da disciplina.

Em suma, a etapa inicial foi de levantamento bibliográfico, concentrado em fontes de engenharia de software. A partir de então, a segunda etapa, foi a escrita dos resumos e fichamentos que auxiliaram na construção do tema da pesquisa alinhado com os objetivos do presente projeto.

6.2 DELIMITAÇÃO E POPULAÇÃO DA PESQUISA

No âmbito do desenvolvimento de software, a discussão que propõe como se deve organizar e funcionar times que constroem sistemas, possui dois momentos distintos. O que antecede a engenharia de software e, o momento a partir da própria engenharia de software, que sistematiza as discussões e oferece novos recursos para as melhores práticas de construção de sistemas.

Neste sentido, o escopo deste trabalho é sobre a organização de times de desenvolvimento de software.

Portanto, através da análise e entendimento sobre organização e funcionamento de times, nos dois momentos acima referidos, temos como objetivo elencar e tabular os conceitos que devem ser levados em consideração na formação de equipes de desenvolvimento.

6.3 ETAPAS DA PESQUISA

A primeira etapa da pesquisa foi o levantamento bibliográfico que aborda engenharia de software para identificar o porquê ela foi necessária nos anos de 1960. Uma vez identificada, buscamos entender quais fatores ocorreram para nascer em um momento chamado ‘crise do software’.

Neste sentido, algumas referências foram encontradas, como os autores Frederick Phillips Brooks e Melvin E. Conway. Em Brooks identificamos uma preocupação inicial e um modelo sistematizado de formação de equipes de desenvolvimento e, em Conway, a abordagem sociotécnica da relação entre estrutura de comunicação da organização com o software produzido como emulação desta relação. A chamada lei de Conway.

A partir de então, o que tornou-se core da pesquisa foi a Lei de Conway e sua versão reversa que, em um segundo momento, também motivou outro levantamento bibliográfico baseado nestes conceitos. Encontramos, portanto, o livro *Team Topologies* de Skelton e Pais (2019) que se fundamenta na perspectiva sociotécnica.

Após a leitura destes temas, iniciamos a etapa escrita do trabalho através de fichamentos que no último momento foram utilizados para compor a discussão teórica do trabalho de conclusão.

De acordo com o que foi sugerido na fase da defesa do projeto de conclusão de curso, incluímos a perspectiva de equipes de desenvolvimento distribuídas para ampliar a abordagem e alinhá-la com uma perspectiva mais atual. Isto posto, outro levantamento bibliográfico foi realizado até encontrarmos o conceito de *remote-first* e os trabalhos dos autores Skelton e Pais (2022) *Remote Team Interactions Workbook* e os níveis do trabalho remoto de Matt Mullenweg (2020).

Por fim, a pesquisa por ser bibliográfica e exploratória, foi finalmente composta por discussão teórica com intuito de analisar os conceitos que contribuem para a montagem de equipes de desenvolvimento.

7. DESENVOLVIMENTO DO TRABALHO, APRESENTAÇÃO DE RESULTADOS

A partir dos conceitos extraídos através desta pesquisa, este trabalho resultou em uma lista de sugestão que pode ser utilizada como referência de conceitos importantes para montagem, análise e reflexão sobre a construção de equipes de desenvolvimento de sistemas. Cada definição dos conceitos, traz abaixo algumas questões que foram formuladas a partir do

que foi desenvolvido ao longo do trabalho. Indica, também, as respostas decorrentes das análises do material teórico utilizado neste trabalho.

CARGA COGNITIVA

É a dimensão do sistema que se espera que um time possa lidar. No sentido psicológico, é a quantidade total de esforço mental usado pela memória no trabalho. Frente a isto, para o bom desempenho do time, a carga mental não deve ser alta.

QUESTÃO

A carga cognitiva da equipe está condizente com sua capacidade de trabalho e com a demanda exigida? O que tem sido feito para diminuir a carga cognitiva da equipe? A organização dos times leva em consideração a carga cognitiva que pode ser encontrada em um projeto?

ESPAÇOS ONLINES

Ferramentas de comunicação online que são organizadas em grupos e canais para promover a comunicação via chats. Devem ser fáceis e intuitivas de serem utilizadas. No contexto de trabalho remoto, é o principal espaço de comunicação e interação.

QUESTÃO

Os chats estão nomeados de modo que possamos entender as interações entre as equipes? Os espaços onlines estão sendo utilizados para promover a comunicação necessária através de grupos e canais públicos?

LEI DE BROOKS

Como a construção de software é, por natureza, um trabalho sistemático - um exercício de inter-relações complexas - o esforço de comunicação é grande e rapidamente domina a diminuição do tempo de cada tarefa individual que foi estabelecido na sua divisão. A adição de mais homens, portanto, aumenta, e não diminui o cronograma.

QUESTÃO

Existem critérios na composição das equipes de desenvolvimento para além de força de trabalho para conclusão de projeto? Há a consideração que a passagem de conhecimento demanda tempo? Existe um mapeamento lógico para entender o atraso nos projetos para além da quantidade de pessoas e trabalho despendido?

LEI DE CONWAY

A Lei de Conway diz que o desenho de uma organização é projetado sobre a arquitetura do software que ela desenvolve. O sistema é uma cópia da estrutura de comunicação de sua organização.

QUESTÃO

Para além de escolher pessoas para compor o time por especialidades demandadas pelo projeto, é estimulada e promovida formas de interação e facilidade de comunicação dentro da equipe, como aviso de atrasos, impedimentos, pontos de atenção, etc ?

LEI DE CONWAY REVERSA

Lei de Conway Reversa, parte do princípio reverso no qual a estrutura organizacional de uma empresa é ordenada pela arquitetura de seu software.

QUESTÃO

No contexto de arquitetura de microsserviços, a equipe é organizada levando em consideração o gerenciamento da nuvem? As boas práticas de segurança, gestão de custos e baixo acoplamento entre os componentes são levados em consideração como responsabilidade na equipe?

NÚMERO DE DUNBAR

O número de Dunbar define o limite cognitivo teórico do número de pessoas com as quais um indivíduo pode manter relações sociais estáveis, ou seja, uma relação onde o indivíduo

conhece cada membro do grupo e sabe identificar em que relação cada indivíduo se encontra com os outros indivíduos do grupo.

QUESTÃO

Levando em consideração a importância da interação e comunicação, a dimensão da equipe, permite relações produtivas que favorecem o contexto organizacional e individual?

REMOT-FIRST

Modalidade de trabalho que se consolidou após uma alternativa ao trabalho presencial durante a pandemia do COVID-19. Nele, todas ou a maioria das pessoas não trabalham em um escritório centralizado, mas sim distribuídas geograficamente e conectadas pela internet.

QUESTÃO

Em qual nível de trabalho remoto a organização se encontra? Existe uma perspectiva e ações para que avance aos demais níveis para vislumbrar novos resultados positivos?

TEAM API

É uma analogia entre o que se espera de uma boa API que segue as boas práticas e o que se espera da comunicação e interação entre equipe, como exemplo: códigos e artefatos da

equipe, Versionamento, Wiki, Práticas e princípios, Comunicação, Road maps e prioridades. Essa analogia baseia a recomendação para interação entre times remotos, uma vez que a informação, comunicação e interação devem ser explícitas.

QUESTÃO

As equipes explicitam sua área de atuação, seus relacionamentos com outras equipes e no que podem contribuir? São fáceis de acessar sem empecilhos?

TEAM FIRST

Deve ser priorizada equipe acima de escolhas individuais que podem impactar na relação entre os membros. A equipe sempre em primeiro lugar para que possamos evitar atritos e suposições individuais que possam desequilibrar.

QUESTÃO

O pensamento entre repartir conquistas e responsabilidades entre o time é disseminado na equipe? O feedback individual é importante e deve ser aplicado, porém existe a prática de fortalecer a equipe, identificar as fragilidades e ressaltar os objetivos alcançados coletivamente ?

TIPOS FUNDAMENTAIS DE TOPOLOGIAS

- *Stream-aligned team;*

- *Enabling Team;*
- *Complicated-subsystem team;*
- *Platform team.*

QUESTÃO

Dentre os quatro modos de interação apresentados, a maneira atual que equipes trabalham e interagem, podem ser identificadas em algum destes modelos de modo que deixe explícito o propósito da topologia?

MODOS DE INTERAÇÃO

- ***Colaboração****: dois times trabalham juntos por um período específico de tempo dentro de um alvo bem definido.*
- ***X-as-a-Service****: um time fornece algum tipo de serviço e outro time consome algo.*
- ***Facilitador****: um time ajudando outro time a corrigir ou descobrir coisas.*

QUESTÃO

A maneira como as equipes colaboram, caso seja sistematizada de acordo com os modos de interação apresentados, traria mais ganhos ao desenvolvimento de software? A interação atual entre as equipes e membros tem apontado déficits técnicos?

8. CONCLUSÃO

A partir daquilo que foi proposto no presente trabalho - apresentar conceitos que norteiam a formação de equipes de desenvolvimento de software - podemos inferir que a construção de equipes deve ser orientada por conceitos e não unicamente por feelings gerados pela experiência. O propósito de apresentar conceitos e as 'leis', como as de Conway e Brooks que se firmaram na engenharia de software, denota que existe um caminho ou um cuidado que deve ser considerado. Entre os caminhos, apresentamos alguns neste trabalho como uma possível contribuição.

A base bibliográfica que fundamentou este trabalho foram: **Team Topologies**: organizing business and technology teams for fast flow (2019) e **Remote Team Interactions Workbook**: organizing business and technology teams for fast flow (2022). A abordagem sócio-técnica, que envolve a relação entre elementos sociais e técnicos, veio ao encontro da proposição Lei de Conway que por sua vez fundamentou os trabalhos acima de Matthews Skelton e Manuel Pais. A partir destes, elencamos os principais conceitos para topologias e sua aplicação no contexto de trabalho remoto.

Contudo, este é um caminho que chama atenção que a construção de equipes vai além da vontade e da capacidade para a conclusão de um projeto. Envolve conceitos que, entre outros, os elencamos na seção de resultados deste trabalho de conclusão.

Devemos salientar portanto, como nomeou Brooks (2018) em seu capítulo, "Não existe bala de prata", o qual diz: "Não há um único desenvolvimento, seja em tecnologia ou em técnica de gestão, que, por si só, prometa sequer uma ordem de grandeza de melhoria de uma década, seja em produtividade, confiabilidade ou simplicidade" (BROOKS, 2018, P173). Neste sentido, os conceitos devem ser aplicados cada qual à luz de seu contexto, testados e validados.

Por fim, no trabalho de Flores (2013), a autora elencou uma série de leituras sobre formação de equipes e constatou que as equipes têm se organizado dentro de uma perspectiva operacional. Esta constatação foi o mote importante para o contraponto que fizemos sobre a importância de uma abordagem sobre formação de equipes além da perspectiva operacional. Permitiu-nos apresentar os diversos conceitos que podem nortear a formação de equipes de desenvolvimento e sinalizar com a mensagem de que é uma reflexão importante e necessária para entregas de qualidade.

REFERÊNCIAS

ALMEIDA, M. S. . **Elaboração de Projeto, TCC, Dissertação e Tese**: uma abordagem simples, prática e objetiva. 2. ed. São Paulo: Atlas, 2014. v. 1. 82p

BIAZZI JÚNIOR, F. de. O trabalho e as organizações na perspectiva sócio-técnica. **RAE-Revista de Administração de Empresas**, *[S. l.]*, v. 34, n. 1, p. 30–37, 1994. Disponível em: https://bibliotecadigital.fgv.br/ojs/index.php/rae/article/view/38173. Acesso em: 7 jul. 2022.

BRASSCOM. **Demanda de Talentos em TIC e Estratégia** . Relatório de Inteligência e Informação BRI2-2021-007 - v112. Disponível em: <https://brasscom.org.br/demanda-de-talentos-em-tic-e-estrategia-%cf%83-tcem/>. Acessado em 07/2022.

BROOKS, Frederick Phillips. **O mítico homem-mês**: ensaios sobre engenharia de software. Tradução: César Brod. Rio de Janeiro: Elsevier, 2018.

FLORES, Fabiana Figueira Sanches. **EQUIPES DE DESENVOLVIMENTO DE SOFTWARE**: concepção, caracteres, espécies e fatores influentes no desempenho. 2013. 368 f. Dissertação (Doutorado) - Curso de Pós-Graduação em Ciência da Computação, Ufpe, Recife, 2013.

FOWLER, J. Susan. **Microsserviços prontos para a produção.** Construindo sistemas padronizados em uma organização de engenharia de software. Novatec. Tradução: Claudio Adas. São Paulo, 2021.

INMETRICS. **Cloud Computing**: como a pandemia acelerou a implementação. como a pandemia acelerou a implementação. 2020. Disponível em: https://inmetrics.com.br/blog/cloud-computing-como-a-pandemia-acelerou-a-implementacao/ . Acesso em: 05/ 2022.

KANBANIZE. **Kanban Explicado para Iniciantes**. Disponível em: https://kanbanize.com/pt/recursos-kanban/primeiros-passos/o-que-e-kanban Acessado em 09/2022.

KIM, G…[et al.]. **Manual De DevOps:** Como obter agilidade, confiabilidade e segurança em organizações tecnológicas. Tradução de João Tortello. Rio de Janeiro: Alta Books, 2018. 464 p.

ROTTA NETO, João. **O Manifesto Ágil.** Disponível em: <http://paginapessoal.utfpr.edu.br/frufrek/pos-web/p/arquivos/O_manifesto_agil.pdf>. Acessado em 07/2022

METODOLOGIA-MAXWELL, **Metodologia**. PUC - Rio de Janeiro 2005.Disponível em:

https://www.maxwell.vrac.puc-rio.br/7162/7162_4.PDF. Acesso em 06/2022.

MICROSSERVIÇOS: **O que são microsserviços?**. 2022. Disponível em: https://aws.amazon.com/pt/microservices.Acesso em: 27 abr. 2022.

MORAIS, Izabelly Soares D.; ZANIN, Aline. **Engenharia de software**: Grupo A, 2020. 9788595022539. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788595022539/. Acesso em: 23 abr. 2022.

MULLENWEG, Matt. **Distributed Work's Five Levels of Autonomy.** Disponível em: https://ma.tt/2020/04/five-levels-of-autonomy/. Acesso em 09/2022.

NUNES, D. J. **Competências do Engenheiro de Software**. in Computação Brasil - Revista da Sociedade Brasileira de Computação. 2015. <Disponível em: https://www.sbc.org.br/images/flippingbook/computacaobrasil/computa_28/cb2015-03.pdf> Acessado em 05/2022.

SKELTON, Matthew; PAIS, Manuel. **Team Topologies**: organizing business and technology teams for fast flow. Portland: It Revolution, 2019. 216 p.

SKELTON, Matthew; PAIS, Manuel. **Remote Team Interactions Workbook**: organizing business and technology teams for fast flow. Portland: It Revolution, 2022. 86p.

SUTHERLAND, Jeff. **SCRUM:** a arte de fazer o dobro do trabalho na metade do tempo. Tradução: Nila da Lua. Rio de Janeiro: Editora Sextante. 2019.

THE DEPARTMENT of Computer Science, University of Cape Town. **Software Engineering.** Disponível em: https://www.cs.uct.ac.za/mit_notes/software/htmls/index.html. Acesso em: 18 maio 2022.

VALENTE, Marco Tulio. **Engenharia de Software Moderna**: princípios e práticas para desenvolvimento de software com produtividade, 2020. 408 p.