UMA AVALIAÇÃO QUALITATIVA DE ESTILOS ARQUITETURAIS A PARTIR DAS CARACTERÍSTICAS DA ARQUITETURA EVOLUTIVA

Agner Esteves Ballejo[1]; Wilson Vendramel[2]

Resumo
A própria evolução dos sistemas de software implica em mudanças constantes, desse modo, se faz necessário adotar padrões e técnicas que facilitem a manutenção. Por um lado, as modificações envolvendo requisitos não funcionais influenciam a arquitetura do software; por outro lado, uma arquitetura torna mais fácil o raciocínio e a gestão de mudanças. A arquitetura evolutiva possibilita que mudanças sejam implementadas de modo mais fácil e sustentável. O objetivo deste trabalho é realizar uma avaliação qualitativa de estilos arquiteturais a partir de características da arquitetura evolutiva. Para tal, foi realizada uma pesquisa com profissionais de desenvolvimento de software a fim de avaliar se as técnicas estudadas estão aderentes à prática.
Palavras-chave: Evolução de software; Estilos arquiteturais; Arquitetura evolutiva; Quantum arquitetural; Acoplamento apropriado; Mudanças incrementais; Mudanças guiadas.

Abstract
The very evolution of software systems implies constant changes, so it is necessary to adopt standards and techniques that facilitate maintenance. On the one hand, changes involving nonfunctional requirements influence the architecture of the software; on the other hand, an architecture makes reasoning and change management easier. The evolutionary architecture makes it possible for changes to be implemented more easily and sustainably. The objective of this work is to carry out a qualitative evaluation of architectural styles based on the characteristics of evolutionary architecture. To this end, a survey was conducted with software development professionals in order to assess whether the techniques studied are adherent to the practice.
Keywords: Evolution of software; Architectural styles; Evolutionary architecture; Architectural quantum; Proper coupling; Incremental changes; Guided changes.

1 Introdução

As mudanças em sistemas de software são inevitáveis e constantes, tanto no decorrer do desenvolvimento quanto, principalmente, após a implantação do produto. Tais mudanças podem ser causadas por fatores de ordem técnica, operacional ou de negócio.

A necessidade de mudança contínua é intrínseca à natureza do software, portanto os programas que compõem um sistema de software precisam ser mais flexíveis às alterações; a modificabilidade deve ser mantida ao longo de seu ciclo de vida. Vale destacar que todos os

[1] Pós-graduado em Engenharia de Software pela Unicamp, graduado em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologias-FATEC, câmpus Zona Leste, professional na área de tecnologia da informação. E-mail: agner_esteves@hotmail.com.
[2] Doutorando em Tecnologias da Inteligência e Design Digital pela Pontifícia Universidade Católica de São Paulo-PUC/SP, mestre em Ciência da Computação pelo Instituto de Computação da Universidade Estadual de Campinas-UNICAMP e mestre em Engenharia de Produção pela Universidade Paulista-UNIP, professor da Faculdade de Tecnologia-FATEC, câmpus Zona Leste. E-mail: wilson.vendramel@fatec.sp.gov.

aspectos envolvendo desempenho, capacidade e qualidade em geral não podem ser projetados e incorporados em um software desde o início. Ao invés disso, essas propriedades são alcançadas de forma gradual por meio de mudanças e refinamento evolutivo (LEHMAN, 1980).

As modificações, principalmente as que englobam os requisitos não funcionais, podem afetar a arquitetura geral de um software, não somente seus componentes individuais. Por exemplo, para garantir que os requisitos de desempenho sejam atendidos, será preciso organizar o sistema para minimizar a comunicação entre os componentes (SOMMERVILLE, 2010).

A arquitetura é uma ponte entre os objetivos de negócio (comumente abstratos) e o software resultante (concreto). O caminho de objetivos abstratos para sistemas concretos tende a ser complexo, porém as arquiteturas de software podem ser projetadas, analisadas, documentadas e implementadas mediante a adoção de técnicas que visam apoiar a realização desses objetivos de negócio, sendo assim, a complexidade pode ser tornar tratável BASS *et al*. (2013).

Dentre as razões para uma arquitetura de software ser importante, três de impacto técnico são listados aqui BASS *et al*. (2013): 1) Uma arquitetura inibe ou permite a obtenção dos atributos de qualidade de um software; 2) Muitos aspectos relacionados à qualidade de um software podem ser previstos a partir de estudos de sua arquitetura; 3) Uma arquitetura torna mais fácil o raciocínio e a gestão de mudanças.

Dentre as atividades do processo de design de software, uma delas é a obtenção de uma arquitetura que represente a estrutura global do software, identificando componentes e suas interdependências. Essa arquitetura é construída com base nas informações do domínio de negócio e dos estilos arquiteturais. Embora um design de software seja considerado uma instância de um determinado estilo arquitetural, os elementos e estruturas definidas como parte de uma arquitetura são a raiz de todo o design. Enfim, o design de software começa com uma consideração arquitetural PRESSMAN; MAXIM, (2015).

Os estilos arquiteturais são uma forma promissora de orientar a evolução arquitetural (TAMZALIT; MENS, 2010). Nesse sentido, os estilos arquiteturais com componentes menores, além de responsabilidades e fronteiras bem definidas, possibilitam uma maior flexibilidade às mudanças, sendo assim, é relevante buscar a construção de arquiteturas com componentes de escopo menor. Além disso, há também práticas que aceleram a entrega de software em lotes menores e que propicia uma percepção mais rápida de falhas no decorrer do processo de desenvolvimento, como a Integração Contínua (Continuous Integration - CI) e a Entrega Contínua (Continuous Deployment - CD) FORD *et al*. (2017).

Ford *et al*. (2017) propõem a construção de uma arquitetura, denominada evolutiva, a qual permite mudanças incrementais e guiadas. A arquitetura evolutiva possibilita que as mudanças sejam implementadas de modo mais fácil e sustentável e, consequentemente, um menor impacto ao introduzir ou modificar funcionalidades no sistema, reduzindo assim riscos e custos relacionados à evolução do software. Vale salientar que a arquitetura evolutiva não é um novo estilo arquitetural ou uma determinada metodologia, mas um conjunto de diversas técnicas trabalhando em conjunto para viabilizar mudanças incrementais de forma controlada para se atingir o objetivo do software.

Diante do exposto, é notório que a modificabilidade é um atributo de qualidade requerido para o software, mas ao mesmo tempo, o fato de lidar com previsões de quais mudanças ocorrerão é uma tarefa árdua e de custo elevado, na maior parte dos casos. Desse modo, Ford *et al*. (2017) entendem que é importante construir uma arquitetura sustentável e aberta às modificações, proporcionando uma maior facilidade de manutenção e custos mais baixos. Essa arquitetura deve ser planejada, guiada e protegida para acomodar mudanças sem grandes impactos.

O objetivo deste trabalho é realizar uma avaliação de estilos arquiteturais, a partir de características da arquitetura evolutiva. Para tal, foram realizadas pesquisas bibliográficas que trouxeram embasamento teórico a respeito de determinados estilos. Além do mais, foi realizada uma pesquisa com profissionais de desenvolvimento de software a fim de avaliar se os estilos arquiteturais elencados estão aderentes à aplicabilidade das técnicas estudadas no cotidiano desses profissionais.

Dentre os estudos realizados com esse tema, Cuadrado *et al*. (2017) por meio de um estudo de caso, validaram a aplicação do estilo arquitetural orientado a serviços na transformação de um sistema monolítico e constataram que houve uma redução de acoplamento e uma melhoria na manutenção do código do software, diminuindo os custos da manutenção do mesmo.

Este trabalho está dividido de forma que seja apresentado os estilos arquiteturais alvos do estudo na seção 2, seguido de uma apresentação das características da arquitetura evolutiva na seção 3, logo após na seção 4, é apresentada a pesquisa com os profissionais atuantes no mundo do trabalho que tiveram alguma experiência com os estilos arquiteturais estudados, a qual também inclui uma análise e discussão dos resultados obtidos e, por fim, a conclusão deste estudo.

2 Estilos arquiteturais

Em todo sistema de software, parte fundamental é sua arquitetura, esta que é responsável por descrever como é a organização dos componentes e como as interações entre eles ocorrem. Isto é comumente representado na forma de modelos visuais que representam a estrutura do sistema de software em alto nível de abstração e de forma compreensível. Além disso, esses modelos expõem as decisões arquiteturais, permitindo identificar novos padrões e estilos arquiteturais, promovendo soluções para problemas recorrentes GARLAN *et al*. (1994).

Um estilo arquitetural estabelece uma família de sistemas em termos de um padrão de organização estrutural. De forma mais específica, um estilo arquitetural determina o vocabulário de componentes e conectores que podem ser usados em instâncias desse estilo, contemplando também um conjunto de restrições sobre como eles podem ser combinados (GARLAN; SHAW, 1994).

O propósito de um estilo arquitetural é contemplar questões relacionadas às decisões de design de arquitetura. Devido à estreita relação entre os requisitos não funcionais e a arquitetura, o estilo e a estrutura da arquitetura particular a ser escolhida para um software devem depender dos requisitos não funcionais do sistema (SOMMERVILLE, 2010).

O modelo de requisitos identifica as propriedades do software, inclusive aquelas que tendem a afetar a arquitetura, como integridade, escalabilidade, segurança, performance, disponibilidade entre outras. Os requisitos podem ser descobertos a partir de reuniões entre os stakeholders, tanto da área de negócio quanto da área técnica. Nessas reuniões, pode haver a construção de modelos visuais (diagramas UML, por exemplo) que permitem elevar a compreensão arquitetural do sistema (BOOCH, 2009). A arquitetura faz parte do modelo de design, porém é extraída a partir do modelo de requisitos (PRESSMAN; MAXIM, 2015).

Mediante o conhecimento dos requisitos do sistema, é possível definir um estilo arquitetural a ser utilizado na estrutura do software, o que proporciona uma maior clareza do sistema e sua construção de forma guiada e mais rápida (GARLAN; SHAW, 1994).

Dentre os estilos arquiteturais estudados por Ford *et al*. (2017), o presente trabalho contempla quatro estilos arquiteturais para serem avaliados: a) monolítico; b) microsserviços; c) baseado em eventos; e d) serverless.

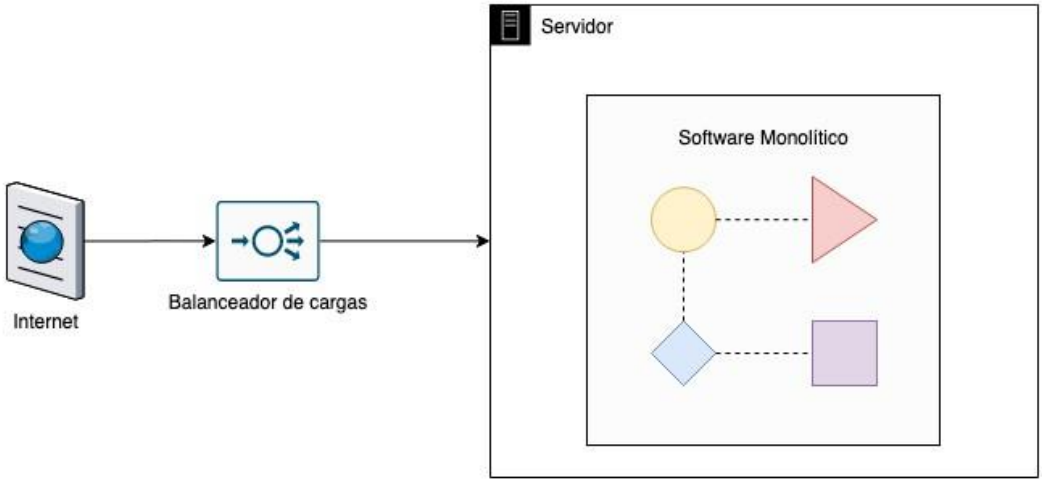
2.1 Estilo monolítico

O estilo arquitetural monolítico é uma abordagem em que toda a aplicação faz parte de um mesmo artefato, portanto, quando há uma alteração em uma pequena parte no código, é

necessário realizar a entrega de todo o código base. Além disso, toda a aplicação é executada sobre o mesmo processo, o qual significa que para escalar horizontalmente, é necessário realizar o deploy de toda a aplicação, ao invés de somente as partes que realmente necessitam de mais recursos (FOWLER; LEWIS, 2014).

A figura 1 mostra uma representação do estilo arquitetural monolítico. Nesse estilo, tanto a aplicação quanto os servidores são preocupações para o desenvolvimento do software. É importante realçar que os módulos (representados por figuras geométricas) estão no mesmo código base e sendo executados no mesmo servidor.

Figura 1 - Estilo arquitetural monolítico



Fonte: Adaptado de Fowler e Lewis (2014).

Vale ressaltar, que para escalar horizontalmente uma aplicação de software monolítica, é necessário ter toda a aplicação sendo executada em servidores diferentes e utilizar um balanceador de carga para balancear o tráfego de requisições entre eles.

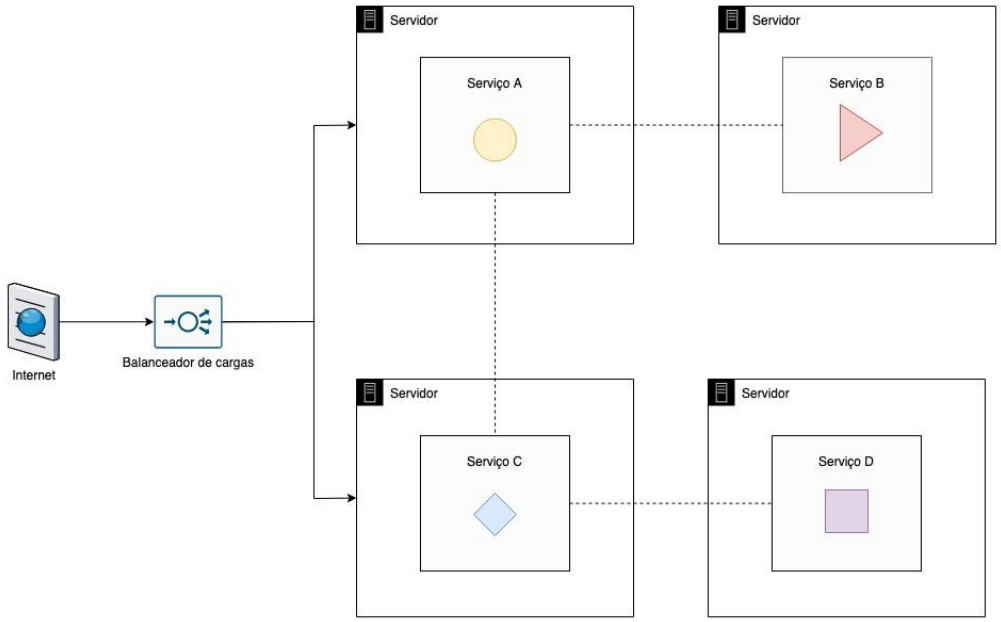
O código base cresce conforme novas funcionalidades são adicionadas, desse modo, tornase cada vez mais difícil saber onde realizar a manutenção ao longo do tempo, pois o código base se torna muito extenso; mesmo que o software esteja modularizado, as dificuldades permanecem (NEWMAN, 2015).

2.2 Estilo de microsserviços

O estilo arquitetural de microsserviços possui uma abordagem de pequenos serviços independentes trabalhando juntos. Cada um desses serviços possui uma responsabilidade e uma fronteira bem definida a respeito do processo de negócio, tornando fácil a localização do código para uma funcionalidade específica. Ao manter o microsserviço com uma fronteira bem definida, o seu crescimento extensivo torna-se mais difícil (NEWMAN, 2015).

A figura 2 exibe uma representação do estilo arquitetural de microsserviços. Os módulos são separados em serviços diferentes (representados pelas figuras geométricas), cada qual com seu próprio código base, e executados em servidores diferentes.

Figura 2 - Estilo arquitetural de microsserviços



Fonte: Adaptado de Fowler e Lewis (2014).

Para escalar horizontalmente uma aplicação de microsserviços, é possível realizar o deploy apenas dos microsserviços que necessitam de mais recursos.

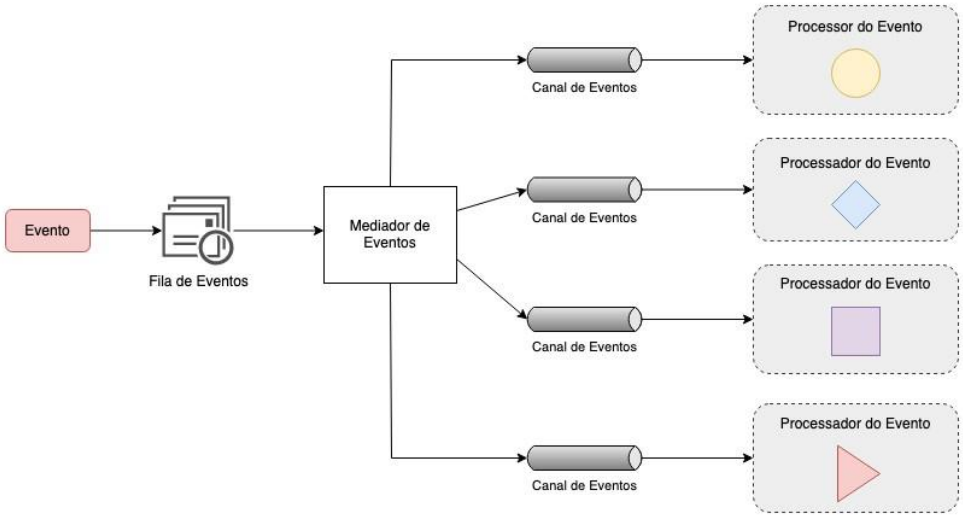
Os microsserviços viabilizam o controle das mudanças em suas aplicações, e tendo em mãos uma ferramenta adequada, essas mudanças vão se tornar frequentes, rápidas e bem controladas (FOWLER; LEWIS, 2014).

2.3 Estilo baseado em eventos

O estilo arquitetural baseado em eventos é uma arquitetura distribuída e assíncrona, comumente utilizada na construção de sistemas de software de alta disponibilidade e escaláveis, pois ela proporciona uma maior flexibilidade de seus componentes; eles são altamente desacoplados, permitindo adicionar ou remover funcionalidades com maior facilidade (RICHARDS, 2015).

A figura 3 expõe uma representação do estilo arquitetural baseado em eventos, utilizando nesse caso, a topologia de mediador. Nessa topologia, há um componente central responsável por orquestrar os componentes que processam os eventos (representados por figuras geométricas), sendo que a chamada a esses componentes ocorre de maneira assíncrona FORD *et al*. (2017). O mediador de eventos e os componentes que os processam podem ter seu código base separado e distribuído em servidores diferentes.

Figura 3 - Estilo arquitetural baseado em eventos.



Fonte: Adaptado de Richards (2015) e Fowler e Lewis (2014).

Em uma aplicação que utiliza uma arquitetura baseada em eventos, é possível escalar horizontalmente apenas aumentando a quantidade de componentes que processam os eventos.

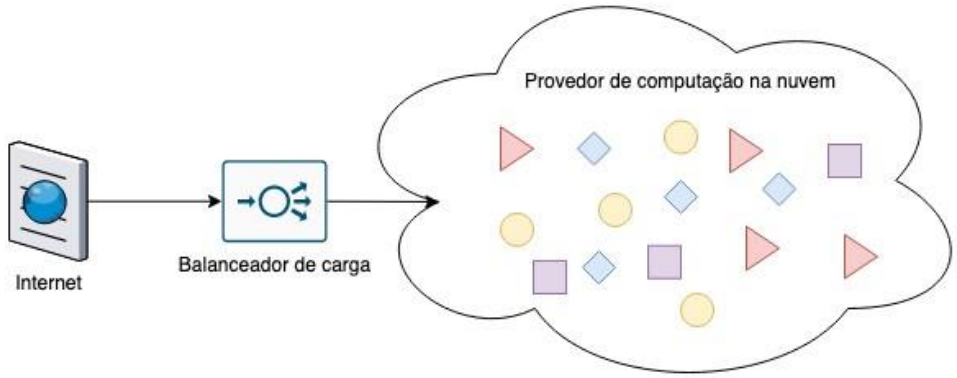
É importante frisar que na topologia de broker, não há um mediador central para realizar a orquestração dos eventos, portanto isso fica a cargo dos componentes que processam os eventos. Por um lado, a mudança fica ainda mais flexível, já que permite a adição de novos processadores de eventos facilmente, porém a mudança, por outro lado, torna-se mais complexa devida à falta de um mediador central, resultando em um tratamento de erros mais difícil FORD *et al*. (2017).

2.4 Estilo serverless

Serverless é um estilo arquitetural, em grande parte por conta de que muitas organizações têm migrado a arquitetura de suas aplicações para containers e microsserviços. Tal modelo viabiliza uma programação simplificada para implantar aplicações em nuvem, que abstrai, se não tudo, grande parte das preocupações operacionais e de recursos de infraestrutura, significando que não há servidores para a aplicação BALDINI *et al*. (2017).

A figura 4 retrata a representação do estilo arquitetural serverless, incluindo funções como serviço (representadas por figuras geométricas), onde cada função pode operar de forma independente de uma das outras em um provedor de computação na nuvem.

Figura 4 - Estilo arquitetural serverless



Fonte: Adaptado de Fowler e Lewis (2014).

Function as a Service (FaaS) é um tipo de computação como serviço, onde a lógica está dividida em funções e, na maior parte dos casos, é executada em resposta aos eventos. Esses eventos, síncronos ou assíncronos, podem ser disparados de fontes externas ao serviço de plataforma de computação na nuvem ou mesmo internamente entre os serviços da plataforma (ROBERTS; CHAPIN, 2017).

3 Características da arquitetura evolutiva

Ford *et al*. (2017) realizaram uma análise e levaram em consideração certas preocupações para cada estilo arquitetural. Em resumo, essas preocupações compuseram três características, sendo elas:

1. Acoplamento apropriado: Como os componentes de um estilo arquitetural estão interligados? Uma mudança em um componente afeta outro?
2. Mudanças incrementais: Como possibilitar a manutenção e/ou introdução de novas funcionalidades de forma sustentável?
3. Mudanças guiadas: Como garantir que o objetivo da arquitetura será mantido?

3.1 Acoplamento apropriado

Acoplamento é a métrica de como os módulos estão interligados em um software. Isto depende da complexidade entre eles, como é o ponto de entrada e o formato de dados compartilhado por eles (PRESSMAN; MAXIM, 2015). Enquanto o baixo acoplamento é bom, o alto acoplamento é ruim, portanto, a busca é pela redução do acoplamento.

Ford *et al*. (2017) introduziram o conceito de quantum, que é a menor parte física existente, logo, o quantum arquitetural é o tamanho dos artefatos que são entregues de forma

independente e com alto nível de coesão; tal granularidade, assim como o baixo acoplamento, são características relevantes para se construir uma arquitetura evolutiva.

Como exemplo, no estilo arquitetural monolítico, o quantum é todo o software, pois existe apenas um artefato a ser entregue. Enquanto no estilo arquitetural de microsserviços, existe uma fronteira e responsabilidades bem definidas entre os serviços dessa arquitetura; eles possuem coesão alta e acoplamento baixo, pois encapsulam a lógica de negócio, viabilizando mudanças incrementais com mais facilidade, além de sua entrega ocorrer de forma independente dos outros componentes, sendo assim, os serviços são o quantum da arquitetura de microsserviços.

3.2 Mudanças incrementais

Arquitetura evolutiva implica em mudanças incrementais na introdução de novas funcionalidades. Essas mudanças devem ocorrer em lotes pequenos para se fazer uso de entrega contínua e mitigar os riscos de implantação FORD *et al*. (2017). Contudo, planejar mudanças incrementais pode ser custoso, já que é preciso prever: a) o que pode mudar; b) a probabilidade da mudança; c) quando a mudança é feita e quem a faz; e d) o custo da mudança BASS *et al*. (2013).

Novas técnicas foram estudadas e desenvolvidas para ajudar no planejamento e implementação de sistemas de software sustentáveis visando suportar tais mudanças.

As principais técnicas são:

a) Reduzir o tamanho dos módulos: a redução do fator quantum de uma arquitetura ajuda a realizar mudanças e a reduzir os custos de desenvolvimento e manutenção;
b) Aumentar a coesão: as responsabilidades de um módulo devem servir ao mesmo propósito, caso contrário, pode-se criar módulos e dividir as responsabilidades corretamente;
c) Diminuir o acoplamento: o baixo acoplamento impede que uma alteração no módulo A, por exemplo, afete o módulo B, não havendo assim efeitos colaterais por causa de uma modificação;
d) Aumentar a cobertura de testes: um software com cobertura alta de testes aumenta a confiança ao realizar as mudanças necessárias, mantendo o objetivo da arquitetura e evitando o surgimento de falhas. Isso é importante para que mudanças sejam realizadas e entregues continuamente em produção sem que o software venha a enfrentar indisponibilidade ou falhas;

e) Entrega contínua: essa prática consiste em criar uma Pipeline - conhecida como fluxo de entrega, ou Deployment Pipeline - o qual monitora alterações no código fonte e processa um fluxo com diversos passos que executam testes de unidade e de 9 integração, constroem os artefatos do software (build) e realizam a entrega desses artefatos em diversos ambientes (teste, homologação, produção ou outros), sendo partes desse processo, tarefas manuais ou automatizadas (FARLEY; HUMBLE, 2011).

3.3 Mudanças guiadas

Certas decisões tomadas são cruciais para o sucesso de um software, portanto é necessário haver um processo confiável e rigoroso para se tomar decisões, mitigar os riscos e maximizar os ganhos FALESSI *et al*. (2011).

Ford *et al*. (2017) incentivam a utilização de uma técnica chamada Fitness Functions (comumente utilizada para determinar "sucesso" em algoritmos genéticos) para guiar o processo de desenvolvimento do software, proteger suas características e verificar o quão perto a arquitetura está de seu objetivo final. Essas são como métricas que auxiliam na detecção quando a introdução de novas funcionalidades compromete uma característica da arquitetura, por exemplo: o atributo de qualidade de segurança é uma característica importante que manipula senhas de usuários, então é definido uma Fitness Function que valida se todas as senhas persistidas no banco de dados estão devidamente criptografas e protegidas.

A definição de Fitness Functions não é uma tarefa simples, porém ela é de extrema importância, a qual deve ser idealizada no planejamento do software, dessa maneira, essa tarefa permite identificar e reduzir os riscos mais cedo e projetar o sistema para mudanças, evitando problemas como tomadas de decisão equivocadas que custam tempo e dinheiro, os quais não viabilizariam a evolução do software de maneira sustentável.

Uma maneira de se construir Fitness Functions é com a criação de testes, que podem ser de unidade, de integração, de aceitação entre outros. Cada modelo de teste possui uma ou mais características, segundo a definição dos autores Ford *et al*. (2017), são elas:

a) Atômica ou Holística: os testes de unidade têm um escopo atômico, enquanto os testes de integração têm um escopo mais abrangente, portanto, holístico;
b) Disparada ou Contínua: testes podem ser executados a partir de um evento, podendo ser, por exemplo, um passo da pipeline de entrega contínua. Já outros testes podem ocorrer de forma contínua (ou constante), por exemplo, com uma transação falsa que ocorre em um ambiente de produção simultaneamente com transações reais;

c) Estático ou Dinâmico: no estático, os testes têm um resultado fixo, como por exemplo, passou ou falhou. Já no dinâmico, o objetivo é que os resultados sejam baseados em um determinado contexto, por exemplo, testes de performance e de escalabilidade podem gerar conflitos entre si, ou seja, quanto maior for a escalabilidade do sistema, menor pode ser sua performance;
d) Automático ou Manual: alguns testes podem ser automáticos, como àqueles disparados por meio de eventos de pipeline. Para outros, pode haver a necessidade de serem processados manualmente devido a sua complexidade, sendo normalmente executados por Quality Assurance;
e) Temporal: testes que ocorrem em um determinado tempo, ou em um intervalo de horas, dias ou meses para validar algo, como por exemplo, se alguma dependência externa possui novas versões;
f) Intencional ou Emergente: alguns testes podem aparecer ao longo do desenvolvimento do software, o que é normal, pois nem tudo pode ser previsto ou descoberto no início;
g) Específico de domínio: os testes podem ser definidos para validar se há novos requisitos regulatórios ou de segurança, de acordo com o domínio de negócio.

4 Aplicação da Pesquisa

Para avaliar os quatro estilos arquiteturais a partir das três características da arquitetura evolutiva estudados neste trabalho, foi realizada uma pesquisa de natureza qualitativa e propósito exploratório, no período de 01 a 15 de maio de 2020. A amostra desta pesquisa é de 44 participantes, os quais representam profissionais que atuam com desenvolvimento de software. O intuito da pesquisa qualitativa é a seleção intencional dos participantes ou dos locais que possam ajudar o pesquisador a compreender o problema ou a questão de pesquisa. Além disso, algumas características da pesquisa de natureza qualitativa são identificadas em Creswell (2010), são elas:

a) pesquisador como um instrumento fundamental; b) significado dos participantes; c) projeto emergente; e d) resultados interpretativos.

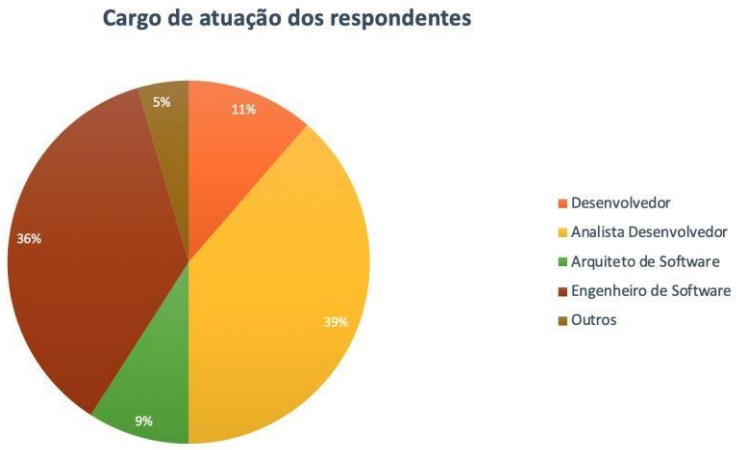
As pesquisas com propósito exploratório procuram identificar o que está ocorrendo e fazer perguntas quando não se sabe o suficiente sobre um dado fenômeno. Um estudo exploratório pode auxiliar na decisão de pesquisar uma determinada questão ou não (GRAY, 2009).

O instrumento de pesquisa adotado foi o questionário do tipo Survey, contendo ao todo 15 questões. As duas primeiras buscam extrair informações sobre a área de atuação e o nível de senioridade dos respondentes; a terceira questão visa descobrir quais estilos arquiteturais cada respondente conhece; as outras 12 estão relacionadas aos quatro estilos arquiteturais e as três características da arquitetura evolutiva. Durante o período da pesquisa, foram enviadas mensagens diretas e e-mails aos participantes da pesquisa.

4.1 Perfil dos respondentes

A primeira questão "**Qual seu cargo atual**", 39% dos respondentes ocupam o cargo de analista desenvolvedor, 36% têm o cargo de engenheiro de software, 11% possuem o cargo de desenvolvedor, 9% têm o cargo de arquiteto de software e, por fim, 5% ocupam outros cargos, como apresentado na figura 5.

Figura 5 - Cargo atual dos respondentes



Fonte: Autores (2021).

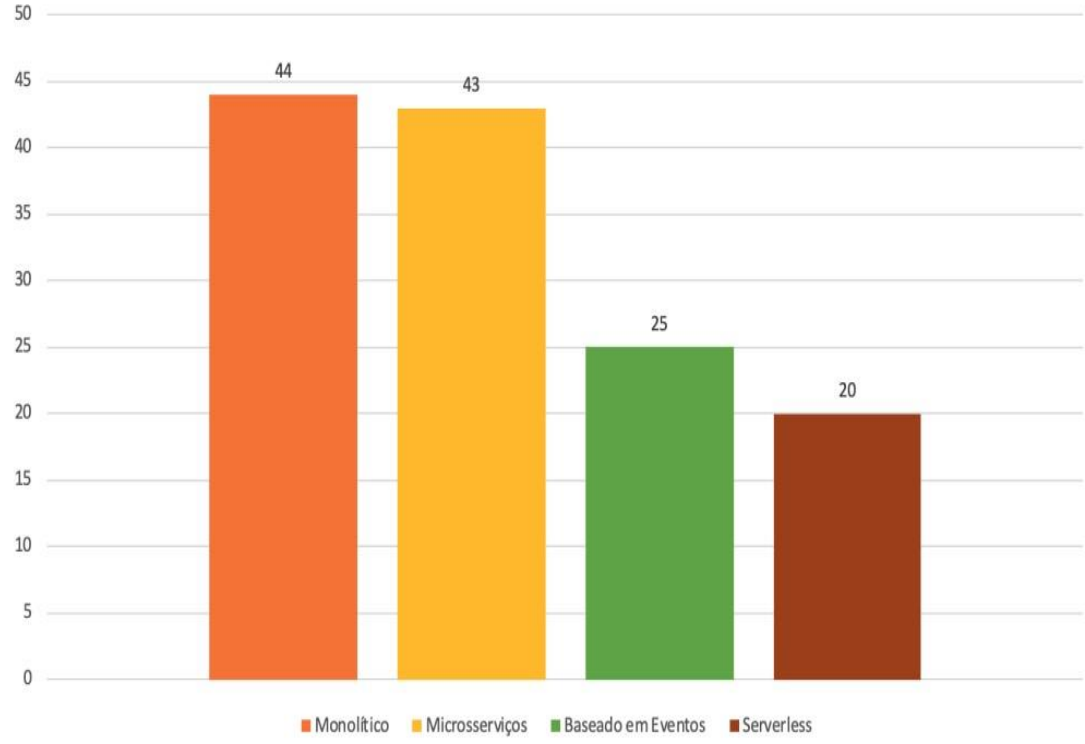
A segunda questão "**Qual seu tempo de experiência atuando no desenvolvimento de software?**", 30% possuem de 3 a 6 anos de experiência, 25% têm de 10 a 20 anos de experiência, 20% possuem de 6 a 10 anos de experiência, 16% têm menos que 3 anos de experiência e, por fim, 9% possuem mais de 20 anos, como apresentado na figura 6.

Figura 6 - Tempo de experiência dos respondentes

Fonte: Autores (2021).

A terceira questão "**Quais estilos arquiteturais você conhece?**", 44 profissionais conhecem o estilo monolítico, 43 conhecem o estilo de microsserviços, 25 conhecem o estilo baseado em eventos e, por fim, 20 conhecem o estilo serverless, como demonstrado na figura 7.

Figura 7 - Estilos arquiteturais conhecidos



Fonte: Autores (2021).

4.2 Análise dos resultados

Após a aplicação da pesquisa, as respostas foram coletadas e analisadas de forma a eliminar os ruídos, uma vez que as respostas para os estilos arquiteturais que os respondentes disseram não ter experiência foram descartadas, permitindo assim haver uma análise mais coerente dos resultados. Este estudo leva em consideração apenas as respostas dos profissionais que possuem alguma experiência com um dado estilo de arquitetura.

O quadro 1 apresenta, em porcentagem, as respostas para as questões 4, 5, 6 e 7 a partir da característica de acoplamento apropriado em relação aos estilos arquiteturais. Além disto, este estudo analisa as respostas com base nos conceitos da arquitetura evolutiva.

Quadro 1 - Análise das respostas para a característica de acoplamento apropriado

Estilo arquitetural	DT	DP	NC/ND	CP	CT	Arquitetura evolutiva	Análise das respostas
Monolítico	20%	50%	14%	16%	0%	Monolítico é um estilo arquitetural cujo código base tende a crescer ao longo de sua existência e, consequentemente, se degrada e não possibilita a evolução sustentável.	A maioria dos respondentes (70%) discorda, no mínimo parcialmente, que o monolítico possui um acoplamento apropriado para a evolução sustentável do software, possivelmente por ser um estilo que se degrada ao longo de sua existência.
Microsserviços	2,3%	0%	4,7%	74,4%	18,6%	Microsserviços possuem um acoplamento de acordo com as integrações entre seus serviços. Se as fronteiras forem bem definidas, o acoplamento será apropriado.	A grande maioria dos respondentes (93%) concorda, no mínimo parcialmente, que microsserviços possuem um acoplamento apropriado me- diante integrações de seus serviços.
Baseado em eventos	0%	12%	8%	68%	12%	Dependendo de como os componentes estão conecta- dos poderá haver um maior acoplamento, sendo que uma abordagem mais descentra lizada pode resultar em um acoplamento mais apropriado à evolução.	A maioria dos respondentes (70%) concorda, no mínimo parcialmente, que o estilo arquitetural baseado em eventos tende a ter um acoplamento apropriado, possivelmente devido a sua descentralização.

Serverless	0%	5%	30%	45%	20%	Serverless é um estilo emergente que abstrai a complexidade da infraestrutura dando um foco maior nas funciona- lidades do negócio, sendo assim, esse estilo possui uma responsabilidade bem defi- nida e seu código base tende a ser pequeno com acoplamento apropriado.	A maior parte dos respondentes (65%) concorda que serverless possui acoplamento apropriado, permitindo entender que é um estilo fácil de evoluir, pois possibilita focar as funcionalidades do negócio.

Fonte: Autores (2021).

O quadro 2 apresenta, em porcentagem, as respostas para as questões 8, 9, 10 e 11 a partir da característica de mudanças incrementais em relação aos estilos arquiteturais. Este estudo também analisa os resultados com base nos conceitos da arquitetura evolutiva.

Quadro 2 - Análise das respostas para a característica de mudanças incrementais.

Estilo arquitetural	DT	DP	NC/ND	CP	CT	Arquitetura evolutiva	Análise das respostas
Monolítico	30%	27%	9%	32%	2%	Dependendo da organização do código do estilo monolítico, as mudanças incrementais poderão ser difíceis, sendo preciso implantar o software por completo, não por partes.	Apesar de o quantum do monolítico ser grande, o mesmo pode ser organizado em módulos que tendem a favorecer as mudanças incrementais. Mesmo assim, 57% dos respondentes discordam que essas mudanças sejam viáveis nesse estilo, provavelmente por não buscarem alternativas para a modularização do software
Microsserviços	0%	0%	0%	44%	55%	Microsserviços possuem fronteiras e responsabilidades bem definidas que proporcionam mudanças incrementais facilmente; uma mudança não deve impactar na outra.	A grande maioria dos respondentes (98%) concorda que microsserviços favorecem mudanças incrementais, muito disso por haver uma consonância com o acoplamento apropriado.
Baseado em eventos	0%	4%	8%	52%	36%	Eventos proporcionam mudanças sem a necessidade de alterar as funcionalidades já existentes devido ao seu acoplamento apropriado, possibilitando a adição mais fácil de novos componentes a fim de se observar os eventos.	A maioria dos respondentes (88%) concorda, no mínimo parcialmente, que o estilo baseado em eventos permite mudanças incrementais, indicando a facilidade de adição de novos componentes.

Serverless	0%	5%	5%	35%	55%	Mudanças incrementais acontecem por meio de deploy do código. Esse estilo se relaciona muito bem com pipelines para lidar com deploy e testes incrementais conforme as mudanças são realizadas.	A grande maioria dos respondentes (90%) concorda que serverless é um estilo o qual viabiliza mudanças incrementais, muito provavelmente por conta de o quantum ser pequeno.

Fonte: Autores (2021).

O quadro 3 apresenta, em porcentagem, as respostas para as questões 12, 13, 14 e 15 a partir da característica de mudanças guiadas em relação aos estilos arquiteturais. Além disto, este estudo inclui uma análise dos resultados com base nos conceitos da arquitetura evolutiva.

Quadro 3 - Análise das respostas para a característica de mudanças guiadas

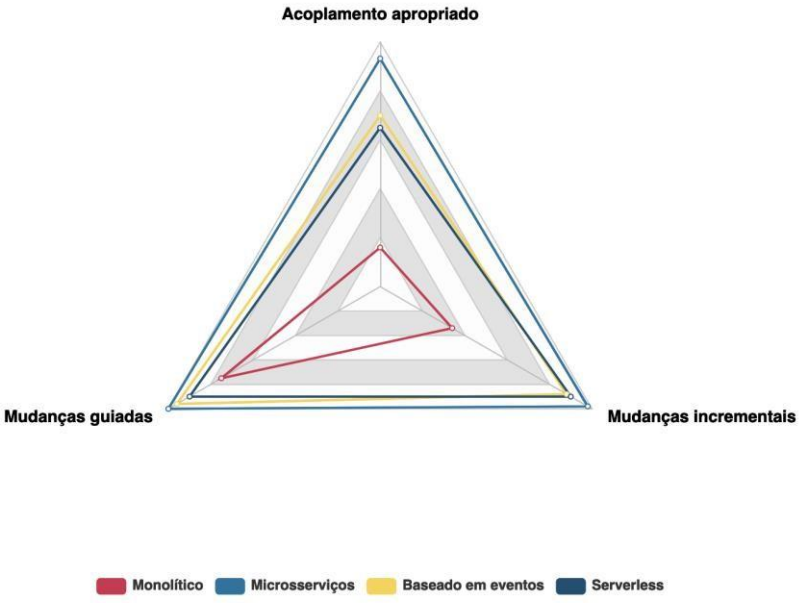
Estilo arquitetural	DT	DP	NC/ND	CP	CT	Arquitetura evolutiva	Análise das respostas
Monolítico	5%	18%	2%	50%	25%	Monolítico é um estilo arquitetural antigo e diversas técnicas foram criadas para propiciar a criação de fitness functions atômicas ou holísticas.	A maioria dos respondentes (75%) concorda, no mínimo parcialmente, que é possível criar fitness functions para o monolítico, isso em grande parte por causa das diversas técnicas e ferramentas criadas para suportar esse estilo.
Microsserviços	0%	0%	0%	53,5%	46,5%	Microsserviços com fronteiras e responsabilidades bem definidas viabilizam a criação de fitness functions atômicas ou holísticas facilmente.	Todos os respondentes entendem, no mínimo parcialmente, que microsserviços possibilitam a criação de fitness functions para guiar as mudanças para um caminho mais sustentável.
Baseado em eventos	0%	0%	4%	76%	20%	As fitness functions atômicas são facilmente implementadas, enquanto as holísticas podem ser complicadas devido ao fator de descentralização dos eventos e a distribuição dos componentes.	Apesar de possíveis dificuldades nas fitness functions holísticas, a grande maioria dos respondentes (96%) entende que é factível guiar as mudanças por meio de fitness functions.
Serverless	0%	5%	5%	45%	45%	Por conta das próprias características da arquitetura distribuída, fitness functions são extremamente importantes e comuns, apesar de não serem triviais em alguns casos.	A grande maioria dos respondentes (90%) concorda que serverless possibilita mudanças guiadas por intermédio de fitness functions atômicas e holísticas.

Fonte: Autores (2021).

4.3. Discussão dos resultados

Após a análise das respostas e visando discutir os resultados obtidos, um gráfico de radar (figura 8) foi elaborado com o objetivo de apresentar a aderência de cada estilo arquitetural para as três características da arquitetura evolutiva abordadas neste estudo. Para todos os estilos, as respostas "Concordo parcialmente" e "Concordo totalmente" foram somadas para se chegar ao valor de aderência.

Figura 8 - Aderência dos estilos arquiteturais



Fonte: Autores (2021).

Pode-se notar um destaque especial ao estilo de microsserviços, o qual foi unânime em relação às três características, deixando evidente ser um estilo arquitetural que proporciona a evolução do software de forma sustentável ao longo de sua existência.

Muito disso é por conta de as fronteiras dos serviços permitirem um acoplamento apropriado que encapsulam a complexidade de seu código e proporcionam mudanças incrementais com mais facilidade. Além disso, pelos serviços possuírem um escopo bem definido e limitado, a arquitetura de microsserviços possibilita a construção de fitness functions facilmente, sendo elas atômicas ou holísticas, as quais aumentam a confiança da equipe de desenvolvimento ao realizar uma alteração.

Os estilos arquiteturais baseado em eventos e serverless possuem uma aderência similar, os dois têm um acoplamento apropriado, viabilizando assim as mudanças incrementais e guiadas. Por um lado, no estilo arquitetural baseado em eventos, os componentes são desacoplados e mudanças incrementais nesses componentes são realizadas facilmente por possuir um acoplamento apropriado, mas por outro lado, devido a ter um processamento

assíncrono de mensagens, fitness functions holísticas podem ser difíceis de serem implementadas.

O serverless, por ser um estilo arquitetural cuja complexidade da infraestrutura é abstraída e as funcionalidades do negócio são o foco, a aplicação tende a ter uma responsabilidade bem definida e um código mais enxuto, tais características o possibilita ter um acoplamento mais apropriado às mudanças incrementais e uma maior facilidade para a criação de fitness functions, ainda que a holística não seja tão trivial, é possível.

5. Conclusão

Uma avaliação de estilos arquiteturais é de suma importância para compreender suas vantagens e desvantagens no design e implementação de arquiteturas de software em relação aos objetivos de negócio a serem atingidos.

A arquitetura evolutiva catalogou e avaliou os estilos arquiteturais de acordo com as seguintes características: acoplamento apropriado, mudanças incrementais e mudanças guiadas. Este estudo realizou uma avaliação qualitativa de quatro estilos arquiteturais a partir de três características da arquitetura evolutiva. Para tal, foi realizada uma pesquisa com 44 profissionais de desenvolvimento de software com a finalidade de avaliar se os estilos arquiteturais e as técnicas da arquitetura evolutiva estudadas são aplicáveis no cotidiano desses profissionais.

Com base nos resultados obtidos, foi possível notar que a avaliação dos respondentes vai de encontro com a maior parte dos conceitos da arquitetura evolutiva em relação aos estilos de arquitetura elencados para esta pesquisa.

Em linhas gerais, os resultados indicam que, quanto menor for o quantum de um sistema de software, maior é a possibilidade de se ter um acoplamento apropriado, além de propiciar a realização de mudanças incrementais e a criação de mecanismos, como fitness functions, para guiar as mudanças.

Este trabalho considera que a combinação adequada entre estilos arquiteturais e as técnicas da arquitetura evolutiva permite a construção de arquiteturas sustentáveis e abertas a mudanças, acarretando uma maior facilidade de manutenção e com custos mais baixos ao longo do ciclo de vida útil do sistema.

Como sugestão de trabalho futuro, pode-se analisar, projetar e implementar uma arquitetura a partir de um ou mais estilos arquiteturais e avaliar qualitativamente e/ou quantitativamente se os conceitos e técnicas da arquitetura evolutiva propiciam a construção e a evolução de sistemas de software mais sustentáveis.

Referências

BALDINI , I.; CASTRO, P; CHANG, P.; CHENG, P.; FINK, S.; ISHAKIAN, V.; MITCHELL, N.; MUTHUSAME, V.; RABBAH, R.; SLOMINSKI, A. **Serverless computing: Current trends and open problems.** In Research Advances in Cloud Computing, pages 1–20. Springer, 2017.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice**. Pearson Education, Inc., 3. Ed, 2013.

BOOCH, G. **Architecture as a shared hallucination.** IEEE Software, 27(1):96–96, 2009.

CRESWELL, J. W. **Projeto de pesquisa: métodos qualitativo, quantitativo e misto.** Artmed, 3.
Ed, 2010.

FALESSI, D.; CANTONE, G.; KAZMAN, R.; KRUCHTEN, P. **Decision-making techniques for software architecture design: A comparative survey.** ACM Computing Surveys (CSUR), 43(4):1–28, 2011.

FARLEY, D.; HUMBLE, J. **Continuous Delivery**. Addison-Wesley, 2011.

FORD, N.; PARSONS, R.; KUA, P. **Building evolutionary architectures: support constant change.** O'Reilly Media, Inc, 2017.

FOWLER, M.; LEWIS, J. **Microservices: a definition of this new architectural term**, 2014**.**

GARLAN, D.; ALLEN, R.; OCKERBLOOM, J. **Exploiting style in architectural design environments.** ACM SIGSOFT software engineering notes, 19(5):175–188, 1994.

GARLAN, D.; SHAW, M. **An introduction to software architecture.** In Advances in software engineering and knowledge engineering, pages 1–39. World Scientific, 1994.

GRAY, D. E. **Pesquisa no mundo real.** Artmed, 2. Ed, 2019.

LEHMAN, M. M. **Programs, life cycles, and laws of software evolution.** Proceedings of the IEEE, 68(9):1060–1076, 1980.

NEWMAN, S. **Building Microservices: designing fine-grained system.** O'Reilly Media, Inc, 2015.

PRESSMAN, R. S.; MAXIM, B. **Software engineering: a practitioner's approach.** McGraw-Hill Education, 8. Ed, 2015.

RICHARDS, M. **Software architecture patterns**, volume 4. O'Reilly Media, Inc, 2015.

ROBERTS, M.; CHAPIN, J. **What Is Serverless?** O'Reilly Media, Inc, 2017.

SOMMERVILLE, I. **Software Engineering**. Pearson Education, Inc., 9. Ed, 2010.

TAMZALIT, D.; MENS, T. **Guiding architectural restructuring through architectural styles.** In 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems, pages 69–78. IEEE, 2010.