22 - Fazendo Times Efetivos

2025-12-24 08:00 am

- Arquiteto tem que guiar o time
- Tem que ser colaborativo.

Fronteiras de Time

- Arquiteto tem muita influência sobre o sucesso do desenvolvimento.
- Deve comunicar restrições e desafios com clareza.
    - Tem um ponto ótimo de restrição e direcionamento aos desenvolvedores.
    - Nem restrito demais nem liberdade demais.

Personalidades de Arquitetos

- Três tipos:
    - **Ultra-Controlador:**
        - Tenta controlar todo e cada detalhe do desenvolvimento.
        - Define todos os nomes, tecnologias, tudo.
        - Tenta substituir si próprio o trabalho dos desenvolvedores.
    - **Desconectado (Armchair):**
        - Não liga pros detalhes de implementação.
        - Desconectado do time de desenvolvimento, não atua como mentor.
        - Só desenha diagramas e fim.
        - Desenvolvedores acabam tomando o papel do arquiteto no desenvolvimento.
    - **Efetivo**:
        - Gera as restrições adequadas ao time.
        - Promove mentoria e colaboração.
        - Ajuda o time de desenvolvimento a superar os desafios.

Quanto Controle?

- *Liderança Elástica*
- Nível de controle deve depender das circunstâncias:
    - **Familiaridade do Time:**
        - Quanto time se conhece.
        - Quanto mais de conhecem, menos controle é necessário.
    - **Tamanho de Time:**
        - Quanto maior o time, mais controle é necessário.
    - **Experiência Geral:**
        - Nível de experiência é parecido dentro do time?
        - Quanto mais sênior, menos controle é necessário. Atuar mais como facilitador.
    - **Complexidade do Projeto:**
        - Quanto mais complexo, mais controle é necessário.
    - **Duração do Projeto:**
        - Quanto mais curto, menos controle é necessário.
        - Projeto curto não tem tempo para formalidades e testes extensivos, melhor deixar o time mais livre.

        - Projetos mais longos requerem ajuda do arquiteto para não deixar largar algum senso de urgência.
- Exemplos:
    - *Table 22-1. Scenario 1 example for amount of control*

Factor	Value	Rating	Personality
Team familiarity	New team members	+20	Control freak
Team size	Small (4 members)	-20	Armchair architect
Overall experience	All experienced	-20	Armchair architect
Project complexity	Relatively simple	-20	Armchair architect
Project duration	2 months	-20	Armchair architect
Accumulated score		-60	Armchair architect

    - *Table 22-2. Scenario 2 example for amount of control*

Factor	Value	Rating	Personality
Team familiarity	Know each other well	-20	Armchair architect
Team size	Large (12 members)	+20	Control freak
Overall experience	Mostly junior	+20	Control freak
Project complexity	High complexity	+20	Control freak
Project duration	6 months	-20	Armchair architect
Accumulated score		-20	Control freak

Sinais de Atenção no Time

- Para definir tamanho ideal de time há três fatores:
    1. *Process Loss:*
        - Lei de Brook
        - Quanto mais pessoas, mais tempo o projeto toma.
        - *Process Loss* é a diferença entre a produtividade teórica máxima e a produtividade real do time.
        - Quantidade de *Merge Conflicts* são sinal de process loss.
        - Paralelizar tarefas é uma boa ideia para minimizar isso.
            - Se não há tarefas paralelas, talvez o time pudesse ser menor.
    2. *Pluralistic Ignorance:*
        - Quanto maior o time, menor a chance de alguém se objetar a alguma decisão.
            - Medo de não ter percebido algum detalhe que todo o resto percebeu (ou finge perceber)
        - Arquiteto deve ser capaz de identificar e evitar que as pessoas só aceitem uma decisão por inércia.
    3. *Diffusion of Responsibility:*
        - Quanto maior o time, pior a comunicação.
        - Incertezas sobre quem faz o quê e tarefas serem abandonadas é um bom sinal de time grande demais.

Aproveitando o Máximo das Checklists

- Checklists são ferramentas poderosas.
    - Usado por cirurgiões e pilotos de avião.
- Arquitetos devem saber quando usar ou não checklists.

- Checklist não deve conter passos dependentes um do outros.
    - (não concordo muito aqui, mas está escrito)
- Também não deve ter tarefas simples demais e conhecidas que já são executadas bem.
- Itens devem ser independentes.
- Não transformar tudo em checklist.
- Têm que ser o menor possível mas ainda significativa com todos os passos necessários.
    - Checklists grandes demais são ignoradas.
- Tudo que pode ser automatizado não deve estar na checklist.
- *Hawthorne Effect:*
    - Pessoas agem diferente quando sabem que podem estar sendo observadas (estejam de fato ou não)
    - Evita que desenvolvedores marquem checklists sem ter concluído de fato.
- Três checklists comuns bastante úteis:
    1. **Checklist de Conclusão de Código pelos Desenvolvedores:**
        - Ter a definição de "Concluído" clara, com uma checklist.
        - Padronização de código e formatação, exceções não-consideradas, padrões do projeto, etc.

Done	Task description
☐	Run code cleanup and code formatting
☐	Execute custom source validation tool
☐	Verify the audit log is written for all updates
☐	Make sure there are no absorbed exceptions
☐	Check for hardcoded values and convert to constants
☐	Verify that only public methods are calling setFailure()
☐	Include @ServiceEntrypoint on service API class

        - *Figure 22-12. Example of a developer code completion checklist*
        - Ferramentas de automação podem facilitar esse processo.
    2. **Checklist de Testes Unitários e Funcionais:**
        - Contém edge-cases.
            - e.g. *checar comportamento para campos faltantes*
        - Costuma ser grande.
    3. **Checklist de Software Release:**
        - Deploy pode conter erros.
        - Mudanças de configuração, adição de bibliotecas externas, updates de database, etc.
        - Cada falha pode virar um item na checklist para um deploy futuro.

Provendo Orientação

- Comunicar design e e guiar as questões principais de implementação.
- Tenta elucidar e fazer as decisões terem justificativa clara.
    - Traz a luz se as decisões e opiniões fazem sentido ou não, tanto técnico quanto de negócio.
- Decisões de usar Frameworks, Bibliotecas Gerais e Bibliotecas Específicas (e.g. leitor de PDF)
    - Quanto mais geral (e.g. framework), mais próximo do arquiteto.
    - Decisões menores não precisam do arquiteto ou desenhando ou aprovando.