Estilos

Temos 3 estilos onde são baseados os testes Saída, Estado e comunicação, para cada estado abordaremos de forma separada seguinte:

- ▼ Baseados em Saída

  O estilo baseados em saída é o estilo mais confiável, em relação a testes, pois e uma testes onde alimentamos a entrada do método testado e avaliamos a sua saída, esse tipo de estilo são aplicáveis apenas para códigos promovem a utilização de programação funcional.

- ▼ Baseados em Estado

  O estilo baseado em Estado, tratasse de após a conclusão do testes, verificar o Estado do sistema ou classes, dependências fora ou compartilhadas, o intuito e verificar onde a operação age para modificar o estado e verificar se esse estado foi mesmo alterado e se é o esperado.

- ▼ Baseados em Comunicação.

  E o de comunicação verifica a comunicação entre o sistema de teste e o colaborados, esse estilo usa da simulação objeto para verificação a comunicação do colaborador. E é o estilo entre os três o menos preciso.

  Uma das características que favorecem esse posição é que para os pilar de regressão, a quantidade de testes realizados, pode criar verificações superficiais, o aconselhado e verificar uma parte de código. Em feedback, por utilizar simulações e as mesma introduzem uma latência são um pouco pior que as outras. A resistência a refatoração, por serem testes que verificam a comunicação estão mais propensos a falsos positivos, e por ultimo manutenção, por todos os detalhes que foram comentados anteriormente, as asserções, simulações, espaço em memoria, esse estilo de teste também tem o maior custo para serem mantidos

Pilar	Baseado em Saída	Baseado em Estado	Baseado em Comunicação
Resistencia a refatoração	Alto	Médio	Médio
Proteção contra regressões	Alto	Alto	Baixo
Manutenibilidades	Alto	Médio	Baixo
Feedback Rápido	Alto	Médio	Médio

Escolas e estilos

A Escola Clássica tem uma preferencia na utilização dos testes baseados em estados e saída, enquanto a Escola Londrina prefere o utilização de testes baseados em saída e comunicação

Paradigma Funcional

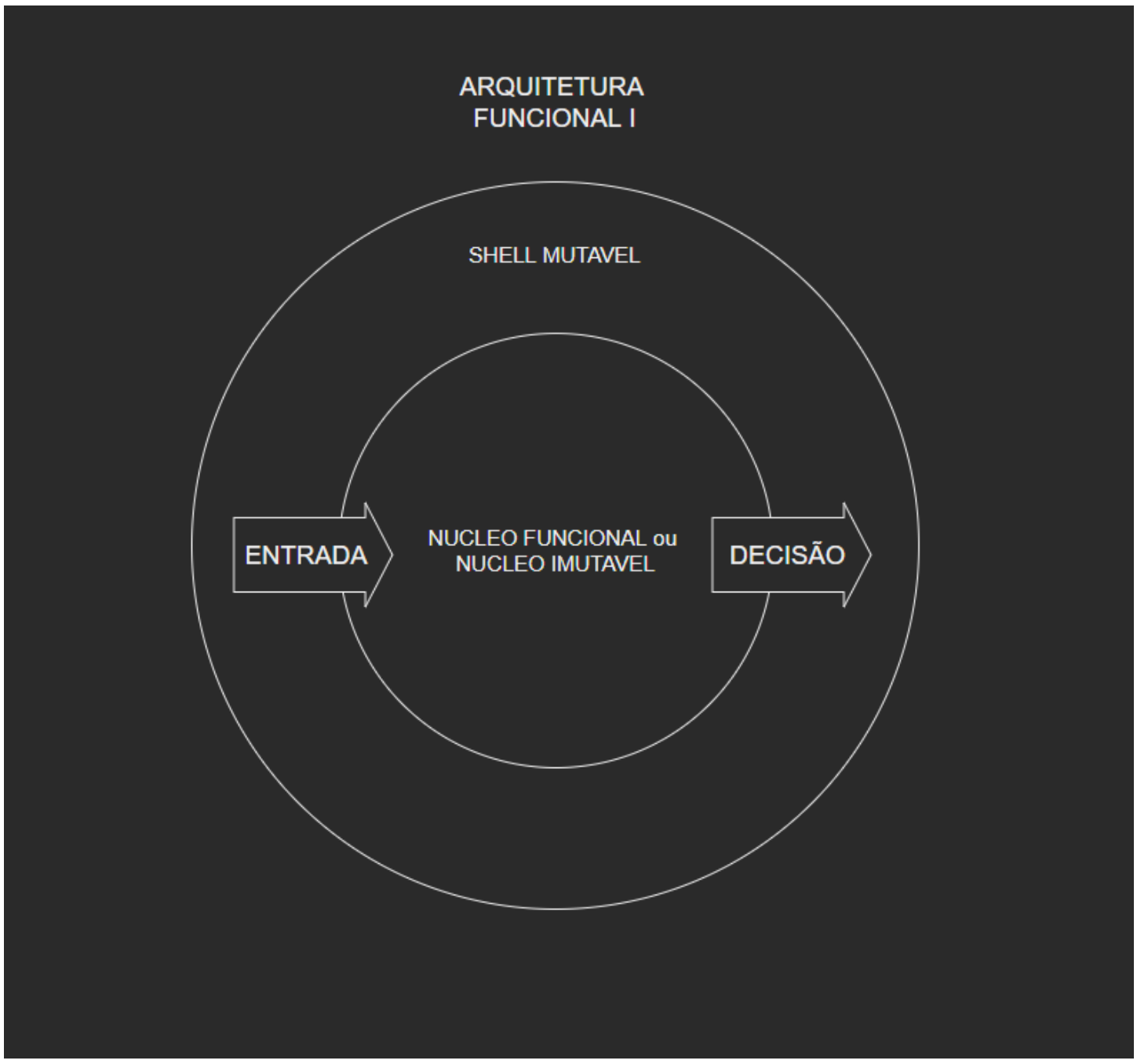
Em programação funcional, e baseada na ideia de funções matemáticas onde, também são conhecidas como funções puras, donde não temos entradas ou saídas implícitas, todas as entrada devem ser explicitadas na assinatura do método. uma de suas características e que para o mesmo conjunto de entradas, sempre teremos a mesma saída, independente da quantidade de vezes que forem executada a função. Os benefícios disso são:

- Mais legíveis e mais testáveis
- Eliminação de Efeitos colaterais, pode ser gerados por entrada ou saídas ocultas dentro do código, podendo gerar resultados inesperados, como mudanças de estado.
- Eliminar Exceções - um exceção pode ser capturada em qualquer lugar da pilha de chamada e pode introduzir um comportamento, uma saída adicional
- Referência a um estado interno e externo - muitos métodos podem não identificar se estão avaliando conceitos externos com bancos de dados ou internos predominantes do próprio sistema. A utilização de data e hora é um exemplo um sistema pode utilizar a referencia de um banco de dados ou de sistema, partir de uma função

Uma boa forma de se avaliar a programação funcional é que simplesmente poderíamos parar executar a chamada e só retornar o valor é o programa continuaria funcionando normalmente. quando esse comportamento e alcançado temos o denominado t**ransparência referencial**

Arquitetura Funcional

A Arquitetura funcional, amplifica a quantidade de código escrito em paradigma funcional. tem por sua vez uma abordagem que separa a logica de negocio com quem a manipula, dessa forma ele empurra a parte complicada como os efeitos colaterais para as bordas da operação de negocio. Dentro da arquitetura funcional temos a representação do que é chamada de núcleo funcional (núcleo imutável), dentro desde núcleo, existe a todas as partes do código que tomam uma decisão. e ele deve ser a parte mais inteligente entre essas duas camadas. Enquanto a outra camada e conhecida como Shell mutável, e essa e a parte mais burra do código, que reage as decisões tomadas, ás tornando visíveis ou efeitos colaterais, também é onde estão todas as entradas.



Um dos objetivos é cobrir a parte do código com o núcleo funcional com testes baseados em saída e o shell mutável com testes de integração.

Arquitetura Funcional vs Hexagonal

Uma das semelhanças entres as arquiteturas e que ambas promovem a segregação de camadas com escopos e funcionalidades, nos dois nos temos uma divisão entre as decisões e ações, na arquitetura hexagonal as decisões são tomadas pelo domínio, onde estão as regras de negocio e serviços de aplicativos as ações. Outra características é a fluxo onde temos no domínio, uma conexão entre as classes onde elas apenas se comunicam entre elas mesmas, elas tem uma autossuficiência. Dentro da arquitetura funcional também temos um isolamento de forma que poderíamos se para o shell mutável do núcleo funcional, e para o shell simular alguns favores simples e como o núcleo não e dependente do shell imutável também poderia funcionar de maneira isolada. Esse e o principal detalhe que faz a arquitetura funcional tão testável.

A diferença entre esses dois esta onde esta os efeitos colaterais, enquanto a arquitetura funcional empurra eles para as bordas, para a área do shell, enquanto na hexagonal ainda á efeitos colaterais feitos pela camada de domínio, e esta tudo bem deste que esses efeitos s[o se restrinjam a camada de domínio.

Desvantagens da arquitetura funcional

O primeiro das desvantagens do paradigma funcional e a perda do desempenho, pois agora o sistema tem que realizar mais chamadas a processos fora do sistema e isso acaba sendo um fator que impacta do desempenho, em alguns sistema a implementação funcional pode apresentar um impacto não perceptível dentro do sistema e nestes casos e melhor optar pela arquitetura funcional. Pois ganhos com ele uma melhor capacidade de manutenção, por tornar o código mais legível e menos descomplicado.

Outras das desvantagens e o aumento de base de código, pois a arquitetura funcional tem uma necessita de uma separação difundida do que é a parte do núcleo funcional e do shell imutável. Mas também em muitos projetos não tem a complexidade necessária para implementação funcional, isso acaba que aumentando a base de código, outros dos fatores que ajudam a aumentar a base de código e a implementação de linguagens voltadas a POO - Programação orientada a objeto, e nestes casos são necessários implementar soluções do estilo baseado em saída como no baseado em estado.