**Relatório - Atividade 6**  
  


Bruno Borges Paschoalinoto 10687472

Gabriel Kenji Godoy Shimanuki 10336719

Gustavo Prieto 4581945

Lucas Rodrigues Cupertino Cardoso 11257543

Otávio Felipe de Freitas 11261249

São Paulo, 2023

**Sumário**

[**1. Introdução**](#_ui6n2qrsxran) **3**

[**2. Categorias de Decisões de Projeto**](#_ay52an3ft4p8) **3**

[**3. Descrição dos conceitos relacionados com testabilidade**](#_t9putry50526) **5**

[**4. Comentários sobre a Lista de Verificação de Projeto para testabilidade**](#_5t00kslfkznt) **6**

[**5. Comentários Relevantes**](#_vd8ejovycrsn) **7**

# **Introdução**

O objetivo do documento é apresentar de maneira mais aprofundada o conceito de *testabilidade* visto na seção teórica da aula. O conceito selecionado pelo grupo possui tamanha importância na indústria que é estimado 30 a 50% do custo de desenvolvimento durante as etapas de verificação do produto.

Testabilidade de *software* – *software testability* – refere-se ao grau em que um artefato de software suporta testes em um determinado contexto de teste. Se a *testabilidade* do artefato de software for alta, é mais fácil encontrar falhas no sistema por meio de testes. Nesse sentido, o presente documento propẽ-se a descrever conceitos relacionados com testabilidade no contexto de processo de tomada de decisões durante o desenvolvimento de um projeto.

# **Categorias de Decisões de Projeto**

Uma arquitetura pode ser encarada como o produto da aplicação de um conjunto de decisões de projeto. Cada decisão corresponde a uma dimensão a qual o(a) arquiteto(a) pode focar sua atenção a fim de resolver os aspectos mais problemáticos de cada categoria [1]. As sete categorias de decisões de projeto são: alocação de responsabilidades, modelo de coordenação, modelo de dados, gerenciamento de recursos, mapeamento de elementos arquiteturais, decisões de tempo vinculativo – *binding time decisions* – e escolha de tecnologias.

Essa classificação não é única, mas é capaz de fornecer uma divisão racional de preocupações a serem contempladas pelo responsável pela arquitetura. Um maior detalhamento das sete classificações é dado a seguir.

* 1. **Alocação de responsabilidades**

Decisões envolvendo o gerenciamento de responsabilidades são, segundo os autores do livro-texto base da atividade, envolvem dois processos principais: identificar responsabilidades importantes do ponto de vista do atributo considerado e determinar a forma pela qual tais responsabilidades são alocadas.

* 1. **Modelo de coordenação**

Corresponde ao mecanismo de interação entre os elementos de *software* segundo lógicas previamente estabelecidas. Decisões referentes ao modelo de coordenação incluem: identificação de elementos pertencentes ou não ao modelo de coordenação, delimitação das propriedades de coordenação (*timeliness*, concorrência, completude, corretude e consistência), e escolha do mecanismo de comunicação – responsável pela execução das propriedades previamente citadas.

* 1. **Modelo de dados**

Corresponde ao conjunto de representações usadas para representar os artefatos de um sistema e a forma de interpretação dos mesmos. Definição de um conjunto de abstrações de dados e seus atributos, compilação de metadados visando consistência e organização dos dados são decisões relacionadas ao modelo de dados.

* 1. **Gerenciamento de recursos**

É o conjunto de decisões sobre o uso de recursos compartilhados na arquitetura do projeto. Tais decisões podem ser a identificação dos recursos a serem utilizados, bem como as limitações de cada um, determinar qual subsistema irá gerenciar cada recurso, determinar como os recursos serão compartilhados, assim como métodos de solução de disputas, e determinar os efeitos e consequências da latência e saturação de cada recurso no projeto como um todo.

* 1. **Mapeamento de elementos arquiteturais**

Existem dois tipos de mapeamento. O primeiro é um mapeamento entre elementos em diversos tipos de estruturas de arquitetura. O segundo é o mapeamento entre os elementos de *software* e os elementos de ambiente (como exemplo, mapeamento de processos para associar as CPUs em que esses processos serão executados). Além disso, mapeamentos podem incluir o mapeamento de módulos e elementos de tempo de execução, atribuição de elementos de tempo de execução aos processadores, a atribuição de itens no modelo de dados para armazenamento e mapeamento de módulos e elementos de tempo de execução para unidades de entrega.

* 1. **Decisões de tempo de vinculação**

Decisões de tempo de vinculação determinam o grau de variabilidade de aspectos de projeto como o tempo de design e *runtime*. Tais decisões impactam fatores como a flexibilidade e aplicação do design, a negociabilidade de protocolos de tempo de execução, e a incorporação de sistemas periféricos durante um mesmo *runtime*. Ao realizar esse tipo de decisão, é necessário considerar custos de implementação da decisão, bem como os custos de modificação depois de realizar a implementação, ou seja, em linhas gerais o custo de oportunidade.

* 1. **Escolha de tecnologias**

Escolhas de tecnologia devem, eventualmente, serem realizadas a partir de uma tecnologia específica. Por vezes a escolha da tecnologia é realizada por terceiros, antes do começo do projeto de arquitetura. Nesse caso, a tecnologia se torna uma limitação/requisito de decisão em cada uma das sete categorias descritas nesta seção. Em outros casos, a arquitetura deve escrever a tecnologia adequada para realizar decisões em cada categoria. A escolha da decisão de tecnologia envolve a decisão de quais tecnologias estão disponíveis para a realização das decisões feitas em cada categoria, determinar se as ferramentas disponíveis para suportar as escolhas de tecnologia são adequadas para o desenvolvimento proposto, determinar a extensão da familiaridade interna, bem como o grau de suporte externo disponível para a tecnologia, determinar os riscos da escolha da tecnologia, como um modelo de coordenação necessária ou limitação da gestão de recursos, e determinação se a nova tecnologia é compatível com a tecnologia já utilizada pelo sistema.

# **Descrição dos conceitos relacionados com testabilidade**

O modelo de teste consiste em um programa que recebe entradas e suas saídas em conjunto com seu estado interno são analisados por um oráculo. O oráculo é um agente – humano ou mecânico – cuja responsabilidade é comparar os dados coletados na saída com o intuito de verificar a corretude em relação à especificação.

Um sistema testável requer a capacidade de controlar cada entrada – sendo desejável o controle sobre o estado interno – para observar as saídas e, no caso de testes automatizados, é utilizado uma *framework* para realizar os testes automatizados – *i.e.* *test harness*. Esse *framework* fornece assistência em diferentes formas com a finalidade de executar os procedimentos e gravar as saídas, ela pode ser totalmente destacada do programa a ser testado, possuindo sua própria arquitetura, requisitos, *stakeholders* e direitos.

Os testes podem ser elaborados por vários desenvolvedores, usuários ou profissionais de *quality assurance* focando uma porção ou o sistema em sua totalidade. As métricas de respostas da testabilidade consistem na eficácia da localização de falhas e na rapidez em que o cenário de testes é executado para o nível de cobertura desejado. Ademais, os casos de testes podem ser uma parte da aceitação do produto ou conduzir o desenvolvimento – *e.g.* metodologia ágil.

Especificando-se alguns pontos dos conceitos relacionados com *testabilidade*, temos:

* **Alocação de responsabilidades**: execução de rotinas de teste e captura de resultados, *log* de atividades que resultaram em falhas ou comportamentos inesperados, controle e observação de estados sistemas que são relevantes para cenários de teste. De maneira geral, busca-se garantir a alocação de funcionalidades com alta coesão, baixo acoplamento, separação de responsabilidades e complexidade estrutural menor possível.
* **Modelo de coordenação**: suporte à execução de módulos de testes e captura de resultados, captura de falhas na comunicação interna aos artefatos do sistema, injeção e monitoramento dos canais de comunicação usados para teste e não introdução de lógicas não determinísticas desnecessárias.
* **Modelo de dados**: garantia de captura dos valores de instâncias de dados, recriação de estados com falhas por meio da injeção de dados, persistência, inicialização, manipulação e criação de abstrações de dados.
* **Mapeamento de elementos arquiteturais**: determinação da forma de testar as possíveis mapeamentos de processos, *threads* e módulos de componentes a fim de obter a resposta desejada e identificar eventuais condições de corrida (*race conditions*).
* **Gerenciamento de recursos**: garantir a suficiência de recursos disponíveis para a execução de testes e captura de resultados. Garantir a recriação de ambiente de produção para realização dos testes de modo a identificar eventuais limites de recursos, captura detalhada do uso para análise em caso de falha, virtualização de recursos para teste etc.
* **Decisões em tempo de vinculação**: assegurar que os componentes gerados em tempo de compilação possam ser testados e que a gama completa de possibilidades de *bindings* possa ser testada.
* **Escolha de tecnologia**: determinar quais tecnologias estão disponíveis para recriação dos cenários de testes, injeção de falhas, repetição de testes condizente com o sistema desenvolvido.

# **Comentários sobre a Lista de Verificação de Projeto para testabilidade**

É notável a presença de planejamento em todos os itens presentes na lista de verificação. Cada ponto apresentado requer uma estratégia prévia para descrever e analisar a especificação do programa, além de fornecer estratégias para alocar e gerenciar os recursos técnicos e humanos.

Sobre os principais aspectos de implementação de testes de *software*, é importante que a alocação de responsabilidades seja bem definida, com funções bem separadas e com baixo acoplamento, a fim de garantir a efetividade dos testes. O modelo de coordenação, por sua vez, deve garantir a comunicação interna adequada dos artefatos do sistema e evitar a introdução de lógicas desnecessárias. O modelo de dados deve permitir a captura e manipulação adequada das informações relevantes para os testes, e o mapeamento de elementos arquiteturais deve ser realizado de forma cuidadosa, a fim de identificar eventuais condições de corrida. O gerenciamento de recursos é um aspecto crítico para a realização de testes, e a escolha de tecnologias adequadas é fundamental para garantir a eficiência do processo. Em resumo, é importante que seja realizada uma abordagem bem estruturada e cuidadosa para a realização de testes de software, visando garantir a qualidade e confiabilidade do sistema desenvolvido.

Sob gerenciamento de recursos, salta aos olhos o trecho de garantia de um ambiente representativo de um cenário real no qual o sistema estará presente. Esse reforço é interessante devido a ocorrências de produtos os quais não foram verificados rigorosamente, como foi o caso do foguete Ariane 5 que sofreu um acidente no vôo 501 por conta de uma conversão de ponto flutuante para inteiro com sinal, sendo que o *testbed* para o sistema não era representativo o suficiente de um cenário real para o lançamento.

# **Comentários Relevantes**

Com tantas falhas presentes nos *softwares* atuais – foguete Ariane 5, acidente do carro autônomo da Tesla, dificuldade de classificação de seres humanos de diferentes raças pela IA construída pela Google –, não é zelo excessivo estressar a importância da *testabilidade* de um programa.

Os alunos, como futuros engenheiros, devem compreender a importância do tópico, assim como estudar os conceitos relacionados para desenvolver produtos funcionais e seguros, tanto para os usuários quanto para os outros indivíduos que podem ser impactados indiretamente. Portanto, a atividade foi crucial para ressaltar os pontos a serem considerados durante a construção de um *testbed*.

**Bibliografia**

[1] BASS & KAZMAN. **Software Architecture in Practice**, 2013.

[2] Notas de aula - 17/02/23