UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
BACHARELADO EM ENGENHARIA ELÉTRICA

LUCYELY CARNEIRO DE FREITAS

ANÁLISE DE TÉCNICAS DE TESTES FUNCIONAIS E DE CAIXA-BRANCA APLICADAS EM ECUS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2021

LUCYELY CARNEIRO DE FREITAS

ANÁLISE DE TÉCNICAS DE TESTES FUNCIONAIS E DE CAIXA-BRANCA APLICADAS EM ECUS

Analysis and project of a class G amplifier for audio frequencies

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica, da Universidade Tecnológica Federal do Paraná.
Orientador: Prof. Dr. Max Mauro Dias Santos

PONTA GROSSA

2021

LUCYELY CARNEIRO DE FREITAS

ANÁLISE DE TÉCNICAS DE TESTES FUNCIONAIS E DE CAIXA-BRANCA APLICADAS EM ECUS

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica, da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 30 de agosto de 2021.

_____

Murilo Oliveira Leme
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Cristhiane Gonçalves
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Max Mauro Dias Santos
Doutorado
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2021

Dedico este trabalho à minha mãe, ao meu noivo e aos meus amigos, pelos momentos de ausência.

AGRADECIMENTOS

O presente trabalho não poderia ser finalizado sem a ajuda de diversas pessoas às quais presto meus agradecimentos. Certamente, esses parágrafos não irão abranger a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre estas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

A minha mãe, Elza, e meu noivo, Gabriel, pelo carinho, incentivo e total apoio em todos os momentos da minha vida.

Ao meu orientador, professor Max, que me mostrou os caminhos a serem seguidos e pela confiança depositada.

A todos os meus colegas de curso e amigos que, certamente, levarei para o resto da vida e que ajudaram de forma direta e indireta na realização e conclusão deste trabalho.

A todos os demais que de alguma forma contribuíram para meu crescimento pessoal e profissional.

“A vida é assim: esquenta e esfria, aperta e daí afrouxa, sossega e depois desinquieta. O que ela quer da gente é coragem.” (ROSA, 1956).

RESUMO

CARNEIRO DE FREITAS, Lucyely. **Análise de técnicas de testes funcionais e de caixa-branca aplicadas em ECUs**. 2021. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) — Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2021.

A indústria automotiva está em constante avanço tecnológico, portanto, traz constantemente novos desafios à engenharia. Para que um veículo seja produzido de maneira confiável e segura para seus usuários, existe uma série de regulamentações e protocolos que buscam garantir que isso aconteça, entre eles, alguns que garantem a qualidade dos softwares embarcados em ECUs. Esse trabalho tem o intuito de analisar algumas técnicas de possíveis testes realizados em ECUs embasados na norma ISO 26262 e a fim de garantir alta qualificação ASPICE para as empresas fornecedoras, sendo um trabalho que aborda esses assuntos de maneira teórica. Serão apresentados conceitos da arquitetura eletrônica veicular e do ciclo de desenvolvimento de software automotivo, além de trazer conceitos e práticas a serem consideradas durante a realização de testes em todos os seus níveis. Serão detalhadas as técnicas de testes caixa-branca e testes funcionais, apresentando algumas técnicas que são mandatórias em cada categoria segundo a ISO 26262, trazendo exemplos bastante didáticos a fim de facilitar o entendimento.

Palavras-chave: Teste funcional. Teste caixa-branca. ISO 26262 (Segurança Funcional). Software automotivo.

ABSTRACT

CARNEIRO DE FREITAS, Lucyely. **Analysis of functional and white-box testing techniques applied to ECUs**. 2021. 72 p. Undergraduate Thesis (Bachelor's Degree in Electrical Engineering) — Federal University of Technology — Paraná, Ponta Grossa, 2021.

The automotive industry is in constant technological advancement, therefore, it constantly brings new challenges to engineering. In order for a vehicle to be produced in a reliable and safe way for its users, there are a series of regimentations and protocols that seek to ensure that this happens, including some that guarantee the quality of the software embedded in ECUs. This work aims to analyze some possible test techniques performed on ECUs based on the ISO 26262 standard and in order to ensure high ASPICE qualification for supplier companies, being a work that approaches these subjects in a theoretical way. Concepts of vehicular electronic architecture and the automotive software development cycle will be presented, in addition to bringing concepts and practices to be considered when performing tests at all levels. The white-box and functional testing techniques will be detailed, presenting some techniques that are mandatory in each category according to ISO 26262, bringing very didactic examples in order to facilitate understanding.

Keywords: Functional testing. White-box testing. ISO 26262 (Functional Safety). Automotive software.

LISTA DE ILUSTRAÇÕES

Figura 1 – Curva de complexibilidade ao longo do tempo na eletrônica veicular. 12
Figura 2 – Funcioinamento de uma unidade eletrônica de controle (ECU). . . . 17
Figura 3 – Diagrama de blocos dos sistemas de controle e monitoramento. . . 18
Figura 4 – Módulos conectados ponto a ponto. 18
Figura 5 – Módulos conectados via barramento de comunicação. 19
Figura 6 – Módulos conectados via barramento em um veículo. 20
Figura 7 – Interface CAN. 21
Figura 8 – Subsistemas eletrônicos em um veículo. 22
Figura 9 – Conteúdo médio de um veículo em porcentagem (algumas figuras podem não somar 100% devido a arredondamentos). 23
Figura 10 – Fases do ciclo de desenovlvimento de software. 25
Figura 11 – Fases do ciclo de desenovlvimento de software no modelo em cascata. 26
Figura 12 – Fases do ciclo de desenovlvimento de software no modelo incremental. 27
Figura 13 – Fases do ciclo de desenovlvimento de software no modelo em V na indústria automotiva. 29
Figura 14 – Fluxo de testes baseados em modelos. 34
Figura 15 – Estrutura para determinação da capacidade pelo ASPICE. 37
Figura 16 – Conjuntos de processos definidos pelo ASPICE. 37
Figura 17 – Tipos de técnicas de testes associados aos níveis de testes. 44
Figura 18 – Exemplo de um caso de teste. 47
Figura 19 – Exemplo de um caso de teste de transição de estado. 53
Figura 20 – Exemplo de um teste de cobertura de instrução. 56
Figura 21 – Exemplo de um teste de cobertura de decisão. 57
Figura 22 – Exemplo de um teste de condição modificada e cobertura de decisão. 58
Figura 23 – Crecimento do valor do projeto de acordo com os testes. 63
Fluxograma 1 – Fluxo e ações proposto para testes caixa-branca. 60
Fluxograma 2 – Fluxo e ações proposto para testes funcionais. 62
Quadro 1 – Níveis de capacidade ASPICE. 38
Quadro 2 – Métodos de testes unitários. 40
Quadro 3 – Métodos de testes de integração. 40
Quadro 4 – Métodos de testes de sistema/aceite. 41
Quadro 5 – Exemplo de condições da técnica de Tabela de decisão. 48
Quadro 6 – Exemplo de condições da técnica de particionamento de equiva - Classes do botão. 51
Quadro 7 – Exemplo de condições da técnica de particionamento de equiva - Classes da temperatura. 51
Quadro 8 – Exemplo de um teste de condição modificada e cobertura de decisão. 58
Quadro 9 – Comparação entre testes Caixa-Branca e Caixa-Preta. 64

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

SIGLAS

ASIL	*Automotive Safety Integrity Level*
ASPICE	*Automotive Software Performance Improvement and Capability dEtermination*
AUTOSAR	*AUTomotive Open System ARchitecture*
BP	*Base Practices*
CAN	*Controller Area Network*
CAPL	*Communication Access Programming Language*
CL	*Capability Level*
CT	Caso de Teste
ECU	*Eletronic Control Unit*
GP	*General Practices*
HIL	*Hardware-In-the-Loop*
IPC	*Instrument Panel Cluster*
ISO	*International Organization for Standardization*
ISTQB	*International Software Testing Qualifications Board*
LED	*Light Emitting Diode*
LIN	*Local Interconnect Network*
MBT	*Model-Based Testing*
MC/DC	*Modified Condition/Decision Coverage*
MIL	*Model-In-the-Loop*
MOST	*Media Oriented System Transport*
PA	*Process Attributes*
PAM	*Process Assessment Model*
PC	*Personal Computer*
PRM	*Process Reference Model*
SAE	*Society of Automotive Engineers*
SIL	*Software-In-the-Loop*
UTFPR	Universidade Tecnológica Federal do Paraná
WP	*Work Products*

SUMÁRIO

1	**INTRODUÇÃO**	**11**
1.1	OBJETIVOS	14
1.1.1	Objetivo geral	14
1.1.2	Objetivos específicos	14
1.2	ORGANIZAÇÃO DO TRABALHO	14
2	**ARQUITETURA DA ELETRÔNICA VEICULAR**	**16**
3	**CICLO DE DESENVOLVIMENTO DE SOFTWARE AUTOMOTIVO**	**23**
3.1	MODELO EM CASCAT	25
3.2	MODELO INCREMENTALL	26
3.3	MODELO EM V	27
3.3.1	Níveis de teste ao longo do ciclo de desenvolvimento em V	31
3.3.1.1	Testes unitários	31
3.3.1.2	Testes de integração	32
3.3.1.3	Testes de sistema	32
3.3.1.4	Testes de aceite	33
3.3.2	Testes baseados em modelos (MBT)	33
3.4	REGULAMENTAÇÕES	35
3.4.1	ASPICE	35
3.4.2	ISO 26262 (*FUNCTIONAL SAFETY*)	39
3.4.3	ISTQB	41
4	**TESTES EM ECUS**	**43**
4.1	TÉCNICAS DE TESTES FUNCIONAIS	45
4.1.1	Técnicas de teste caixa-preta	46
4.1.1.1	Teste de tabela de decisão	46
4.1.1.2	Particionamento de equivalência	49
4.1.1.3	Análise de valor limite	51
4.1.1.4	Teste de transição de estado	52
4.1.2	Técnicas de teste baseadas na experiência	54
4.1.2.1	Suposição de erro	54
4.1.2.2	Teste exploratório	54
4.1.2.3	Teste baseado em checklist	55
4.2	TÉCNICAS DE TESTES CAIXA-BRANCA	55
4.2.1	Teste de cobertura de instrução	56
4.2.2	Teste de cobertura de decisão	57
4.2.3	Condição modificada e cobertura de decisão (MC/DC)	58
5	**RESULTADOS E DICUSSÕES**	**59**
6	**CONCLUSÕES**	**65**
	REFERÊNCIAS	**67**
	ÍNDICE REMISSIVO	**72**

1 INTRODUÇÃO

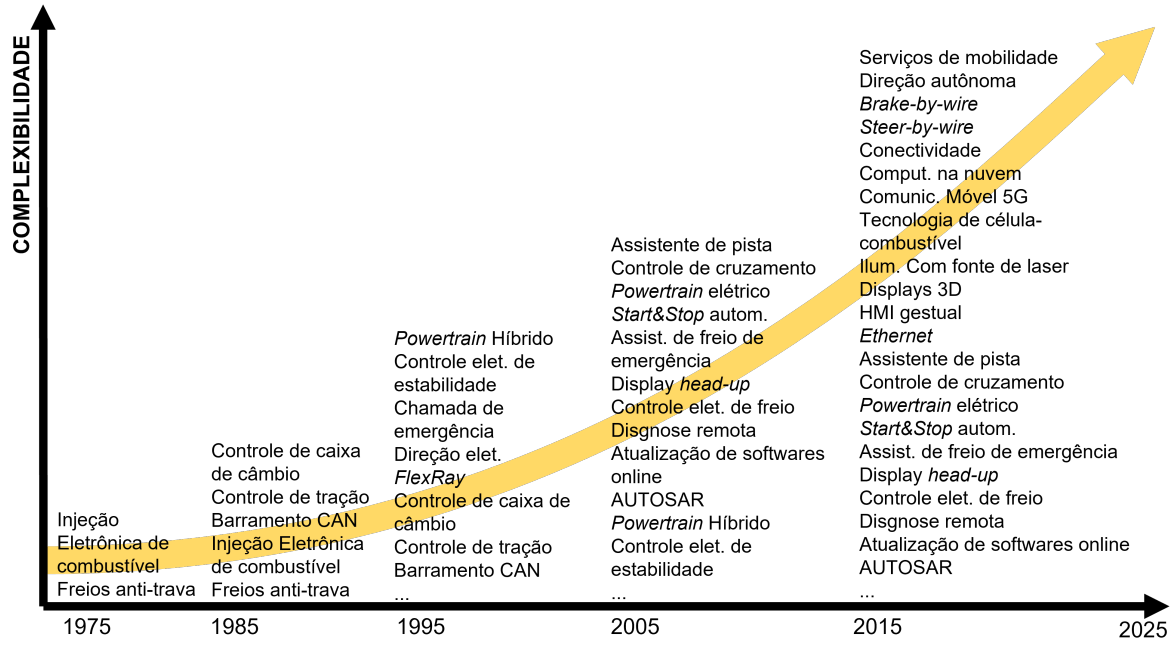
Veículos automotivos são meios de transporte que compõem o sistema rodoviário de mobilidade ao redor de todo o mundo. Ao longo do tempo, a indústria automotiva vem enfrentando novos desafios frente ao alto grau de complexibilidade que o desenvolvimento de veículos requer para atingir padrões de segurança e conforto ao usuário final. Com a implementação de sistemas eletroeletrônicos distribuídos que gerenciam grande parte das funcionalidades presentes nos veículos modernos, estes têm se tornado máquinas cada vez mais complexas e eficientes (AHSAN; STOYANOV; BAILEY, 2016).

A curva de crescente complexibilidade no desenvolvimento da eletrônica apresentada na figura 1, mostra como os elementos foram se complementando aos requisitos para o desenvolvimento de softwares para módulos eletrônicos automotivos, levando em consideração desde a injeção eletrônica de combustível, que foi um dos primeiros componentes eletrônicos inseridos em automóveis, até as funções mais recentes, como assistente de freio de emergência, conectividade, direção autônoma entre outras. Com ela, é possível observar que quanto mais se aumentam as funções requeridas, maior a complexibilidade ao longo do desenvolvimento e implementação destes requisitos, consequentemente, se tornam maiores também os riscos em segurança e responsabilidade (EBERT; FAVARO, 2017).

Com isso, não há apenas um aumento contínuo no número de componentes eletrônicos nos veículos, mas também um grau significativamente mais alto na necessidade de redes de comunicação e um aumento nos volumes de dados, uma vez que a maioria das novas funções automotivas depende da troca de dados entre si. Como resultado, tem-se o desafio em manter a crescente complexidade sob controle e garantir a continuidade da qualidade e confiabilidade das funções. Para isto, a indústria automotiva desenvolveu padrões, como o “AUTOSAR” (*AUTomotive Open System ARchitecture*), que fornecem a clareza necessária no entendimento de um sistema ou função. Padrões de comunicação entre fornecedores, como os sistemas de barramento serial CAN (*Controller Area Network*), LIN (*Local Interconnect Network*), *FlexRay*, MOST (*Media Oriented System Transport*) e *Ethernet* garantem ainda mais clareza nos níveis de comunicação mais baixos (CHRISTMANN, 2017a) entre os equipamentos.

A eletrônica presente em um veículo é representada por várias unidades eletrô-

Figura 1 – Curva de complexibilidade ao longo do tempo na eletrônica veicular.



Fonte: Adaptada de Ebert e Favaro (2017).

nicas de controle (ECU - *Electronic Control Unit*), que se interligam por meio de redes de barramento serial, capazes de fazer a conexão entre todas as partes necessárias para realização de cada função (ONUMA; TERASHIMA; KIYOHARA, 2017). Os veículos atuais, em sua totalidade, possuem várias ECUs para o controle e monitoramento de seus vários subsistemas. Elas ocupam três papéis principais dentro do desenvolvimento do veículo, sendo o primeiro, o controle e gerenciamento das funções, o segundo são as funções de diagnóstico e o terceiro é a supervisão de todas as funções (SCHAUFFELE; ZURAWKA, 2010).

As ECUs são controladores que se baseiam em software embarcado para seu funcionamento, contendo inúmeras funções que são determinadas em seus requisitos de desenvolvimento. Durante o processo de maturação de softwares para aplicação em veículos, grande parte do tempo é empregado a testes para validação de suas aplicações, já que, por abranger tantas funções complexas, requer alta acurácia em sua conferência, pois pode envolver desde questões de conforto ao usuário final, até segurança à sua vida e do meio em que está inserido (AHSAN; STOYANOV; BAILEY, 2016) e para isso, o desenvolvimento de produto segue um processo padrão para a indústria automotiva.

O processo de em que se desenvolvem os softwares para a indústria automotiva

se baseia no modelo V tradicional de desenvolvimento de produto. Este modelo possui foco direcionado às atividades de verificação e validação em relação aos requisitos ao longo de todo o processo. No caso específico do desenvolvimento de softwares automotivos, inicia-se o processo com a especificação dos requisitos necessários para o software, então passa para a etapa de desenvolvimento, para então seguir para a validação do produto, onde são realizados variados testes e conferências com os requisitos inicialmente apresentados em uma ordem hierárquica, abrangendo todo o desenvolvimento do produto. Este modelo também representa os diferentes níveis de refinamento do software, por exemplo, a análise de requisitos e os testes de sistema acontecem no mesmo nível (LIU; ZHANG; ZHU, 2016), estes serão mais bem detalhados no capítulo 3 do presente trabalho.

Para que os testes realizados no processo de desenvolvimento dos softwares possam ser eficazes, é necessário que se sigam alguns padrões que cubram o software por completo, garantindo que chegue ao usuário final um produto seguro, que atenda a todos os requisitos e atenda a todas as regulamentações necessárias. Para que isto ocorra de maneira completa e eficiente, são desenvolvidos métodos específicos que buscam otimizar a maneira como os testes são desenvolvidos e aplicados em cada etapa do processo de desenvolvimento.

Fazendo uma analogia com a construção civil, por exemplo, um engenheiro ou engenheira civil elabora o projeto de um prédio que será construído, após o projeto entregue ao mestre de obras, este então decide os materiais que irá utilizar na construção, mas as diretrizes, as métricas e toda a parte de especificações regulamentares do projeto, já foram definidas pelo engenheiro responsável. No caso de testes de sistemas embarcados automotivos, o profissional de engenharia de sistemas responsável pela elaboração dos testes é como o engenheiro civil, ele é responsável pelo design dos testes, pelas técnicas utilizadas de acordo com as regulamentações e níveis de confiabilidade vigentes, assim como o profissional de civil é responsável pelo desenvolvimento do projeto de construção. Com as técnicas definidas, então o responsável pela execução dos testes pode escolher as ferramentas que irá utilizar para os testes em questão, pode escolher entre variadas ferramentas de testes, linguagens de programação (CAPL, Python, .NET etc.), testes automatizados, etc, assim como o mestre de obras decide se vai utilizar tijolos ou blocos, por exemplo. Dessa forma, é notória a importância do profissional responsável pela elaboração do design dos testes e que ele tenha como

diferenciar as técnicas e as metodologias para abordar as situações que irá enfrentar na indústria.

O presente trabalho tem o intuito de analisar algumas das técnicas utilizadas na elaboração de testes dentro da indústria automotiva e, ao fim, relacionar as técnicas aos níveis de confiabilidade que o software embarcado em uma unidade de controle possa conter.

1.1 OBJETIVOS

1.1.1 Objetivo geral

O objetivo geral deste trabalho é analisar técnicas de testes funcionais e de caixa-branca necessários para a validação de unidades eletrônicas de controle no ciclo de desenvolvimento de produto na indústria automotiva.

1.1.2 Objetivos específicos

Os objetivos específicos são:

- entender como funcionam os testes ao longo do ciclo de desenvolvimento do produto automotivo;
- entender o ciclo de desenvolvimento de software para ECUs, levando em conta duas das principais regulamentações que englobam os testes em ECUs (ASPICE e ISO 26262);
- aprofundar os estudos relacionados a testes funcionais e de caixa-branca no processo de desenvolvimento de produtos na indústria automotiva, já que é uma área de estudos ainda com muito potencial de pesquisa e estudo.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em 6 capítulos, onde o capítulo 2 trata da Arquitetura da eletrônica veicular, onde serão apresentados os conceitos da construção e da comunicação entre os subsistemas encontrados na eletrônica presente em um automóvel.

No capítulo 3, apresenta-se o Ciclo de Desenvolvimento de Software Automotivo. Nele serão apresentados os processos para o desenvolvimento de produto na indústria automotiva, justificando sua utilização e contextualizando nos objetivos deste trabalho.

No capítulo 4, serão abordadas as Técnicas para testes funcionais e caixa-branca em ECUs, onde serão apresentados conceitos a respeito dos testes realizados no ciclo de desenvolvimento apresentado no capítulo 3. Serão detalhadas as principais técnicas utilizadas para os testes atualmente.

No capítulo 5, tem-se os Resultados e discussões, onde são apresentados alguns pontos de discussões e recomendações a respeito de como aplicar esses conceitos formais de testes no domínio de aplicações automotivas levando em consideração requisitos específicos da *Functional Safety* (ISO 26262) no desenvolvimento de unidades eletrônicas.

Por fim, no capítulo 6, tem-se as Conclusões, contendo uma síntese dos conhecimentos adquiridos, detalhe da relevância do trabalho realizado e futuros temas para discussão.

2 ARQUITETURA DA ELETRÔNICA VEICULAR

Em sua maioria, os sistemas elétricos e eletrônicos (E/E) que se encontram nos veículos motorizados dependem uns dos outros, se influenciam e se complementam entre si. Este capítulo apresenta uma explicação de como ocorre a comunicação entre os módulos presentes nos veículos.

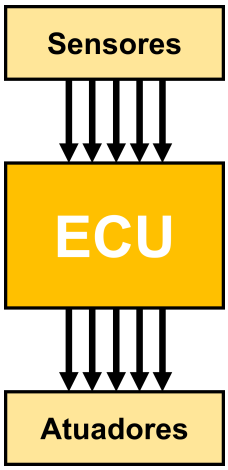
Conforme apresentado por Bosch (2014), a arquitetura eletroeletrônica veicular presente em um carro por volta do ano de 1950 compreendia aproximadamente quarenta cabos apenas, que eram necessários unicamente para os sistemas de bateria, partida, ignição, iluminação e sinalização. A complexidade do cabeamento começou a crescer de forma rápida com os primeiros sistemas eletrônicos de injeção de combustível e ignição. Onde a adição de sensores no motor, por exemplo, sensor de velocidade e sensor de temperatura, tinham a função de enviar sinais para a unidade eletrônica de controle (ECU) do motor, enquanto os injetores de combustível esperavam os sinais de acionamento vindos da mesma ECU. Um aumento adicional na complexidade do cabeamento resultou da introdução e rápida adoção do sistema de freio antitravamento (ABS - *Antilock Braking System*), importante recurso de segurança presente na maioria dos veículos atuais. Enquanto sistemas de conforto e conveniência, como, janelas com vidros elétricos, também viriam a fazer parte do padrão para equipamentos nos carros. Todos esses sistemas requerem cabos de conexão adicionais para conectar sensores, elementos de controle e atuadores à ECU.

Dependendo da classe dos veículos atuais, existem entre 20 e 80 ECUs instaladas (RIBEIRO, 2018), sendo que elas são responsáveis por controlar equipamentos como o motor, o sistema de freio ABS e os airbags, portanto, o número de microcontroladores no veículo tem aumentado continuamente nos últimos anos. Esses sistemas eletrônicos podem ser provenientes de diferentes fabricantes que utilizam interfaces previamente acordadas, embora ainda próprias de cada fornecedor (BOSCH, 2014).

Por definição, uma unidade de controle eletrônico (ECU) é um computador embutido nos veículos para controlar sistemas ou subsistemas mecânicos ou eletrônicos. É um componente eletrônico, que recebe como entrada seus sensores ou outras ECUs e usa atuadores para controlar as funcionalidades do veículo. Por exemplo, o sistema de freios ABS pode receber informações do módulo de controle do powertrain para verificar se o controle de tração está funcionando (ALAM, 2018). A ECU processa

as informações recebidas dos sensores de acordo com procedimentos matemáticos, denominados algoritmos de controle. Os resultados desses cálculos formam a base para os sinais de disparo enviados aos atuadores (BOSCH, 2014), conforme ilustra o diagrama na figura 2.

Figura 2 – Funcioinamento de uma unidade eletrônica de controle (ECU).



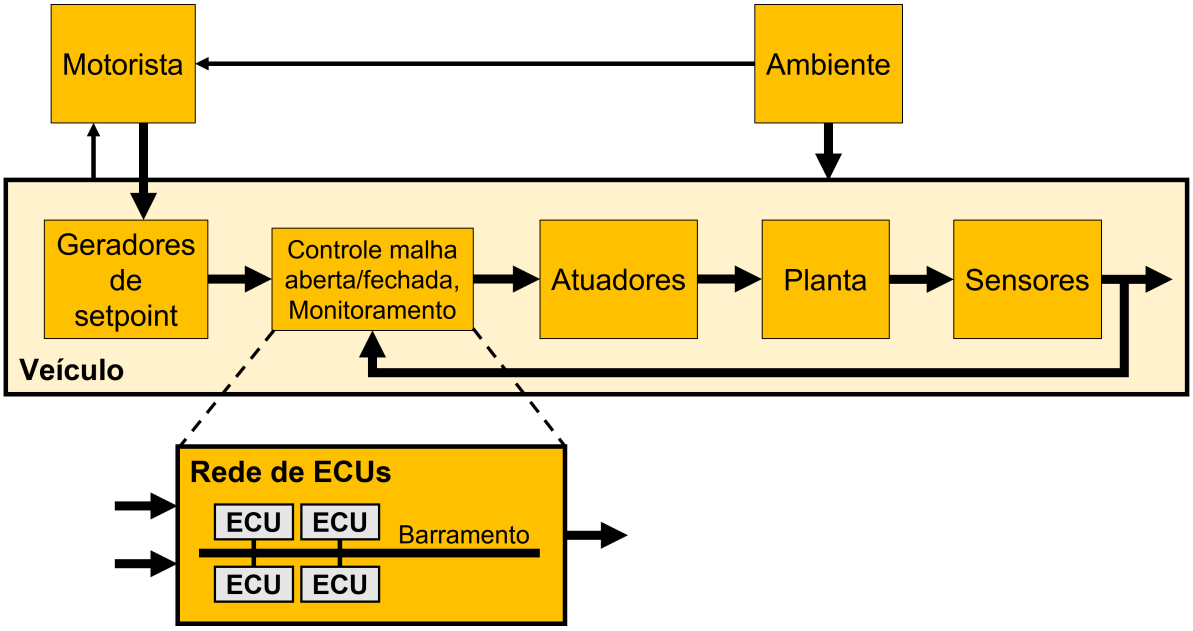
Fonte: Autoria própria.

A configuração típica dos sistemas de controle eletrônico das ECUs e monitoramento do veículo são representadas na figura 3. Geralmente, os seguintes componentes estão envolvidos em tal sistema: geradores de setpoint, sensores, atuadores, ECUs e o sistema controlado em si, chamado de planta. A interconexão entre as redes das ECUs envolvidas facilita a troca de dados entre os módulos. É válido destacar que o motorista e o ambiente influenciam diretamente na maneira como o veículo se comporta (SCHAUFFELE; ZURAWKA, 2010).

Vista por si só, uma ECU representa apenas um meio para um fim, pois é um componente isolado. Apenas um sistema completo compreendendo ECUs, geradores de setpoint, sensores e atuadores irão influenciar e monitorar a planta, ou seja, responder às ações ou solicitações do motorista. No entanto, em muitas situações, e especialmente quando, como é frequentemente o caso, os chamados sistemas embarcados estão em funcionamento, a implementação eletrônica de funções nem mesmo é visível para o usuário do veículo (SCHAUFFELE; ZURAWKA, 2010). Conforme mostrado na figura 3, os sistemas de controle a bordo do veículo podem ser representados como um diagrama de blocos, onde os componentes são apresentados como blocos, com setas representando o fluxo de sinal entre eles.

O crescimento da proporção de componentes eletrônicos em automóveis pode

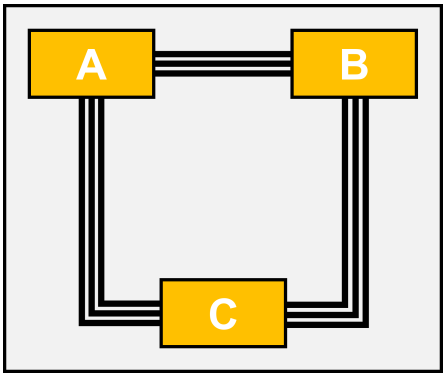
Figura 3 – Diagrama de blocos dos sistemas de controle e monitoramento.



Fonte: Adaptada de Schauffele e Zurawka (2010).

ser atribuído principalmente ao crescimento da microeletrônica e da tecnologia empregada no desenvolvimento de sensores e é anterior ao ano de 1981 (BEREISA, 1983). No princípio, muitos dos novos sistemas eram integrados aos veículos por meio unidades eletrônicas de controle dedicadas por aplicação. Portanto, tornou-se cada vez mais necessária a conexão de cabos entre todas as ECUs para possibilitar a troca de dados por meio de sinais elétricos. Antigamente, essa troca ocorria de maneira separada entre os módulos eletrônicos, representados por “A”, “B” e “C” nos blocos da figura 4.

Figura 4 – Módulos conectados ponto a ponto.

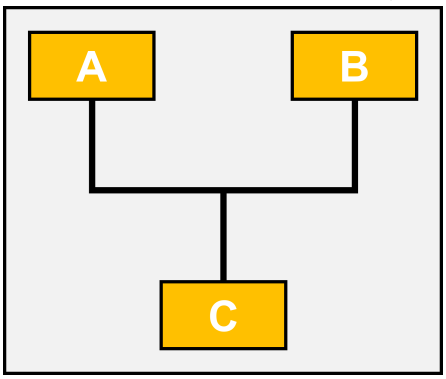


Fonte: Adaptada de Christmann (2017a).

Conforme a quantidade de informações vem crescendo, no entanto, os chicotes de cabos se tornavam tão grandes que seu peso e variedade de conectores se tornaram problemáticos, além dos desafios logísticos que surgiram durante a fabricação, manutenção e desenvolvimento desse nicho de produtos (ABBOTT-MCCUNE; SHAY,

2016). Uma solução para todos os problemas mencionados foi a introdução do chamado barramento de comunicação. Os muitos cabos individuais foram substituídos por um único cabo que é compartilhado por todas as informações de todas as ECUs, assim como ilustra a figura 5.

Figura 5 – Módulos conectados via barramento de comunicação.



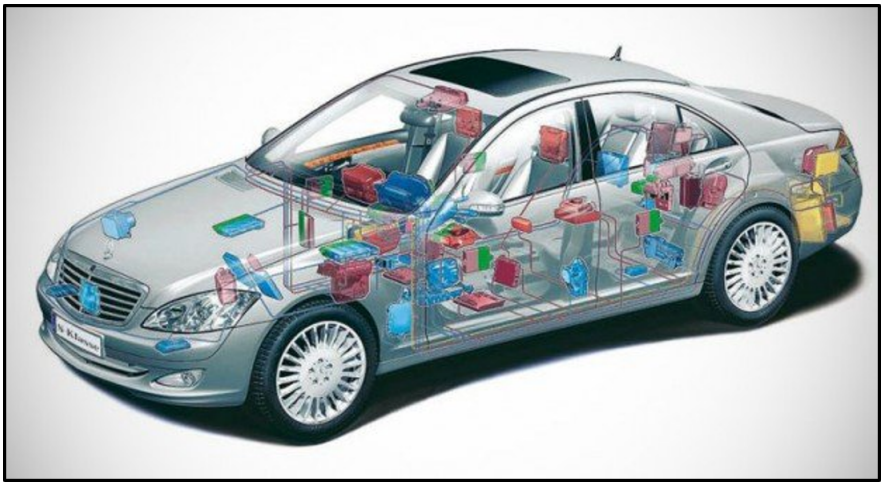
Fonte: Adaptada de Christmann (2017a).

Contudo, por se tratar de uma quantidade massiva de dados, foi necessário encontrar maneiras de organizar a transmissão dessas informações através de recursos comuns na indústria, o que seria então solucionado com a utilização de sistemas de barramento serial. Uma característica comum a todos os barramentos é que cada ECU conectada compartilha uma única entrada e saída e, ao contrário das redes, a informação não precisa passar pelas ECUs. Assim, quando uma ECU envia informações em um barramento, todas as outras ECUs recebem as informações quase simultaneamente. A condição “quase” se dá por conta do tempo de propagação do sinal no cobre, que é de aproximadamente 5 ns/m, e pelo tempo de propagação do sinal nos circuitos de interfaceamento de comunicação CAN (CHRISTMANN, 2017a).

O barramento CAN (*Controller Area Network*) foi desenvolvido pela Bosch (2014), é um sistema de barramento de dados poderoso e se tornou o padrão para sistemas de rede em veículos motorizados. A linha de dados do barramento CAN permite que as ECUs troquem informações específicas e relevantes entre si. Inicialmente, a rede era composta por apenas algumas ECUs, como o sistema de gestão do motor, o controle eletrônico de estabilidade e o controle da transmissão. Gradualmente, outros sistemas expandiram essa rede, especialmente nas áreas de conforto, conveniência e conectividade. Hoje, é o padrão para a comunicação entre ECUs em diferentes áreas da eletrônica veicular (sistema de transmissão, suspensão, eletrônica da carroceria e conectividade) e forma um poderoso *backbone* para interligar essas áreas em rede.

Sistemas de barramento adicionais (por exemplo, barramento LIN, sistema em anel MOST) são usados para transmissão em altas taxas de dados com requisitos de tempo real comparativamente baixos no veículo (CHRISTMANN, 2017a). A figura 6 mostra a conexão eletrônica entre diversos módulos (ECUs) de um veículo através do barramento CAN.

Figura 6 – Módulos conectados via barramento em um veículo.



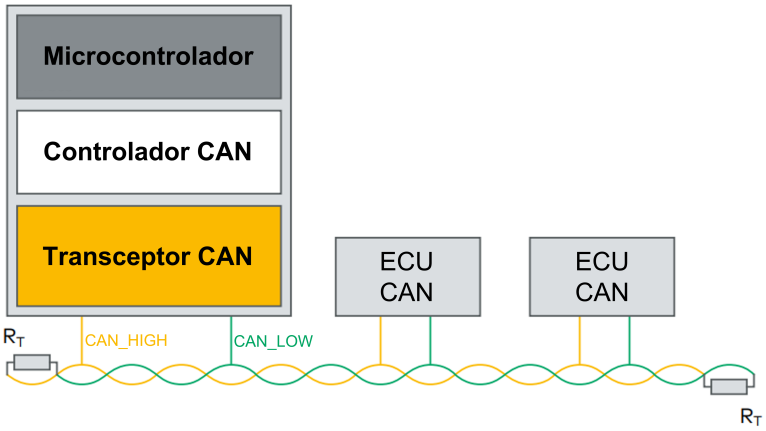
Fonte: Adaptada de Cortesini (2017).

De acordo com Christmann (2017b), o protocolo CAN funciona usando a forma de endereçamento seletivo do receptor da mensagem. O identificador (ID) indica o conteúdo dos dados transmitidos, mas não o destino. Assim, uma mensagem pode ser recebida e avaliada por todas as ECUs no barramento (distribuição de mensagem). A aplicação em uma ECU receptora decide então se avalia a mensagem recebida por ela. A ECU pode até definir um filtro de aceitação em seu controlador CAN ao iniciar, o que oculta mensagens CAN desnecessárias no fluxo de mensagens com base em seus IDs. O protocolo CAN oferece dois tamanhos de identificadores: 11 bits e 29 bits. O identificador de 11 bits (formato padrão), também conhecido como CAN 2.0A ou CAN *Standard*, é predominantemente usado em veículos de passeio. Ele possibilita 2.048 mensagens diferentes, enquanto o identificador de 29 bits, também conhecido como CAN 2.0B ou CAN *Extended*, oferece 536.870.912 mensagens. Este último é utilizado principalmente em veículos comerciais para protocolos de software baseados em CAN, como o SAE J1939, mas também pode ser encontrado em veículos de passeio. O endereçamento seletivo do receptor oferece algumas vantagens, como a economia de custos por meio do uso compartilhado de sensores por todas as ECUs no barramento, a fácil implementação de funções distribuídas, assim como permite

diferentes configurações sem adaptação de hardware ou software.

A interface de comunicação CAN (BOSCH, 2014) consiste em um controlador CAN e um transceptor CAN (figura 7). O controlador CAN opera com o protocolo CAN, nesse componente ocorre a conclusão das mensagens, acontece o controle propriamente dito de acesso ao barramento, a transmissão e recepção de mensagens e a temporização de bits (CHRISTMANN, 2017b). O transceptor CAN conecta o controlador CAN fisicamente a ambos os fios CAN (CAN_HIGH e CAN_LOW) e é nele que ocorre a tradução dos bits em níveis de tensão (transmissão de mensagens) e a amostragem dos níveis de tensão e encaminhamento ao controlador (recepção de mensagens). O software do microcontrolador presente tem o papel de comunicar as ECUs presentes na rede por meio das mensagens no barramento.

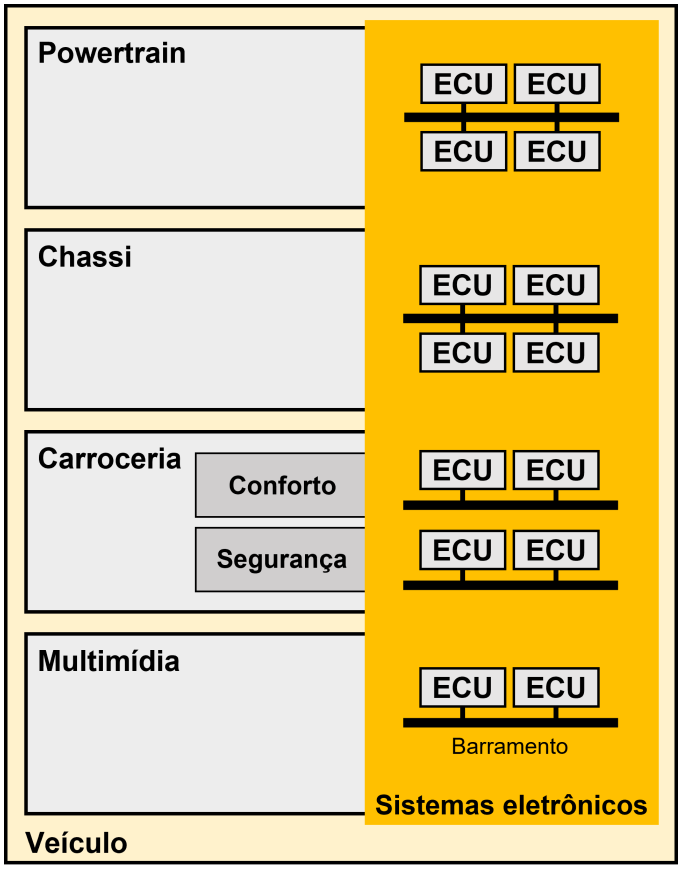
Figura 7 – Interface CAN.



Fonte: Adaptada de Christmann (2017b).

Com a definição dos principais conceitos que compõem a arquitetura eletrônica veicular, pode-se então dividir todos os sistemas eletrônicos dentro de um automóvel em alguns subsistemas. No desenvolvimento de veículos, a expressão "*Divide et impera!*" (SCHAUFFELE; ZURAWKA, 2010) refere-se ao método de particionar o veículo nos subsistemas de powertrain, chassi, carroceria e multimídia, figura 8. Seguindo um procedimento passo a passo, esses subsistemas são divididos em subsistemas e componentes secundários. Os componentes de software desses sistemas são desenvolvidos separadamente, mas em paralelo e são testados ao final de cada etapa do processo. Os componentes são subsequentemente avaliados e integrados em subsistemas nos vários níveis do sistema. Na etapa final, o powertrain, o chassi, a carroceria e os subsistemas multimídia são integrados para então criar o veículo.

Figura 8 – Subsistemas eletrônicos em um veículo.



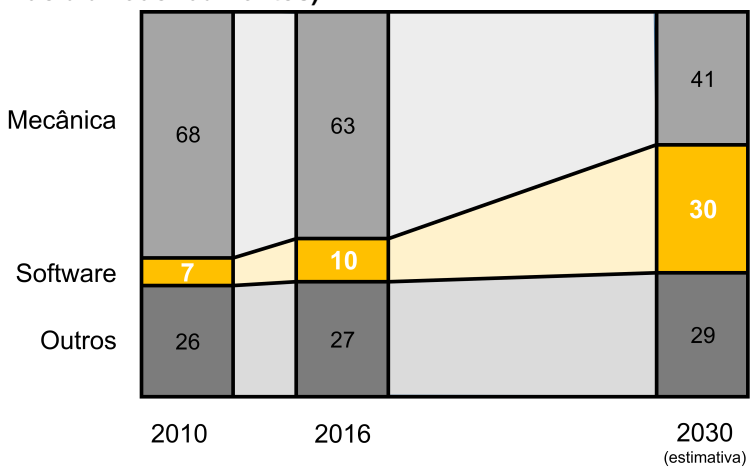
Fonte: Adaptada de Schauffele e Zurawka (2010).

A abordagem para correto funcionamento do fluxo de desenvolvimento dos componentes de software que compõem o veículo não só requer uma divisão de trabalho bem definida no desenvolvimento de subsistemas e componentes, mas também exige que as equipes de desenvolvimento cooperem quando forem tomadas decisões sobre como particionar e posteriormente integrar os sistemas em termos de espaço de instalação, funções do veículo, e tecnologia de produção, requisitos legislativos etc. Além disso, o desenvolvimento de subsistemas e componentes de veículos geralmente é realizado por meio de estreita cooperação entre fabricantes e fornecedores de veículos.

3 CICLO DE DESENVOLVIMENTO DE SOFTWARE AUTOMOTIVO

De acordo com Aboagye et al. (2017), estima-se que 30% do valor de um veículo em 2030 será referente a software embarcado nas ECUs. A figura 9 apresenta essa informação em comparação com a proporção de software embarcado nos anos de 2010 e 2016.

Figura 9 – Conteúdo médio de um veículo em porcentagem (algumas figuras podem não somar 100% devido a arredondamentos).



Fonte: Adaptada de Aboagye et al. (2017).

Então, pode-se observar que para validação de todas essas funções de software embarcado nos veículos, a fim de garantir os requisitos de segurança, conforto, e regulamentações, a necessidade de testes que cubram todas as funções, com metodologias específicas é notável.

O principal produto para a composição eletrônica dos veículos são as ECUs, sendo que, de acordo com ASPICE (2018), 85% das funções dos veículos são controladas por software, nelas estão contidas todas as informações necessárias para controle e correto funcionamento do automóvel, portanto o processo para seu desenvolvimento é bastante complexo e composto por várias etapas, desde planejamento até a entrega final. Este capítulo trata da apresentação do ciclo de desenvolvimento de produto utilizado para a elaboração de software embarcado em uma ECU, apresentando as etapas do ciclo e algumas regulamentações vigentes para o correto desenvolvimento de produto.

Pode-se definir ciclo de desenvolvimento de software, como um método pelo qual o software é elaborado de maneira sistemática e que aumenta a probabilidade de

conclusão do projeto de software dentro do prazo e mantém a qualidade do produto conforme os padrões e regulamentações. Para o desenvolvimento eficaz de um projeto de software, é necessário que se siga uma sequência de atividades que se distribuem entre os desenvolvedores envolvidos no projeto (MISHRA; DUBEY, 2013). Qualquer processo de desenvolvimento de software é dividido em vários estágios lógicos que permitem a uma empresa de desenvolvimento, organizar seu trabalho de forma eficiente para construir um produto com a funcionalidade necessária dentro de um prazo e orçamento específicos. Todos os projetos de software passam pelas fases de definição de requisitos, análise de recursos, design do sistema, implementação e testes de garantia de qualidade (KLOPPER; GRUNER; KOURIE, 2007).

Empregar um modelo de desenvolvimento de software é uma questão de escolha em nível gerencial, dependente do gestor responsável pela equipe de desenvolvimento e das regulamentações associadas ao projeto em questão. Sendo que dentre os variados modelos de desenvolvimento, cada um possui suas vantagens e desvantagens, e cada modelo pode fornecer melhores funcionalidades em uma situação do que em outra. Com isso, o desafio é definir qual modelo deve ser selecionado para fornecer um determinado conjunto de funcionalidades levando em consideração circunstâncias específicas. Um modelo de ciclo de desenvolvimento, teoricamente, pode atender a condições particulares e, ao mesmo tempo, outro modelo também pode parecer adequado aos requisitos, mas deve-se considerar a compensação ao decidir qual modelo escolher (PRESSMAN, 2010).

Um modelo de ciclo de desenvolvimento de software é dividido em atividades distintas e específicas, tem como objetivo determinar a forma como estas atividades são organizadas e todo o esforço requerido para o desenvolvimento do software. Em resposta às abordagens tradicionais de desenvolvimento, novas metodologias surgiram com o passar dos anos (FOWLER, 2000).

Segundo Mishra e Dubey (2013), uma alta porção dos esforços necessários no desenvolvimento de software não tem processo definido e pode ser melhor descrita como sendo uma atividade caótica que fica entre “codificar e corrigir”.

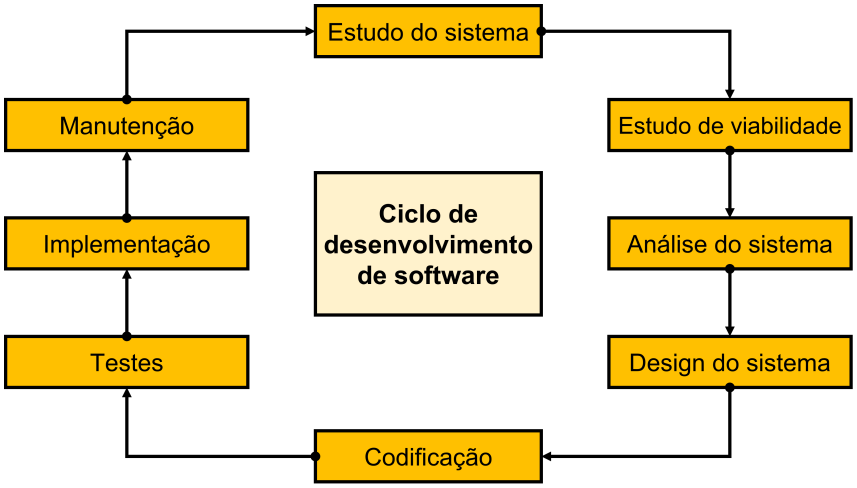
Em um modelo de ciclo de desenvolvimento, normalmente, as etapas que estão presentes são as seguintes:

- compreender o problema (por meio da coleta de requisitos);
- decidir um plano para uma solução (projetar);

- codificar a solução planejada (implementação);
- testar o software desenvolvido; e
- dar manutenção ao produto.

As fases comuns de um ciclo de desenvolvimento de software presentes em todos os modelos podem ser representadas pelo diagrama da figura 10.

Figura 10 – Fases do ciclo de desenovlvimento de software.



Fonte: Adaptada de Mishra e Dubey (2013).

Como é possível observar na figura 10, além das atividades realizadas durante o desenvolvimento do software, a manutenção ainda é realizada após a conclusão do desenvolvimento principal. O software precisa ser mantido em atualização, pois, com frequência, existem alguns erros residuais que permanecem no sistema e devem ser removidos, à medida com que são observados na sequência do desenvolvimento, a manutenção é inevitável para sistemas de software (RUPARELIA, 2010).

A seguir, serão detalhados alguns modelos de ciclos de desenvolvimento de software que deram origem ao atual modelo utilizado em projetos automotivos, o modelo em V, também detalhado no decorrer deste capítulo.

3.1 MODELO EM CASCAT

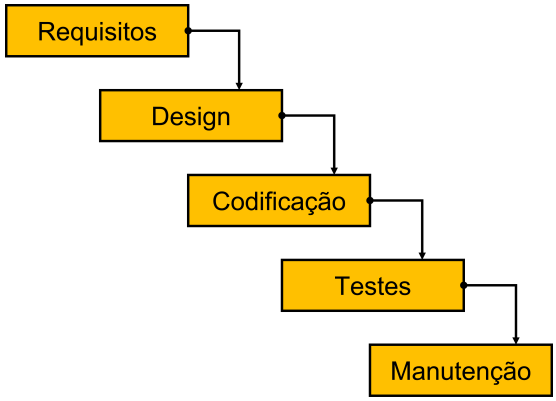
O modelo em cascata foi proposto por Royce (1970), é o modelo clássico de engenharia de software. Este modelo é um dos mais antigos e é amplamente utilizado em projetos governamentais e em muitas empresas e pode ser definido como um modelo sequencial linear. Como esse modelo enfatiza o planejamento nos estágios iniciais, ele abrange as falhas de design antes que elas se desenvolvam. Além disso, sua

documentação e planejamento intensivos fazem com que funcione bem para projetos nos quais o controle de qualidade é uma grande preocupação.

O modelo começa com o estabelecimento de requisitos de sistema e requisitos de software e continua com design de arquitetura, design detalhado, codificação, testes e manutenção, de tal maneira que a fase não se repita e o desenvolvimento não avance para a próxima fase até, e a menos que, a fase anterior esteja completamente concluída. Entretanto, não é muito aplicável quando os requisitos do projeto são dinâmicos e podem sofrer variações no decorrer das fases do projeto (BHUVANESWARI; PRABAHARAN, 2013).

As etapas do modelo de desenvolvimento de software em formato de cascata são apresentadas na figura 11.

Figura 11 – Fases do ciclo de desenovlvimento de software no modelo em cascata.



Fonte: Adaptada de Mishra e Dubey (2013).

3.2 MODELO INCREMENTALL

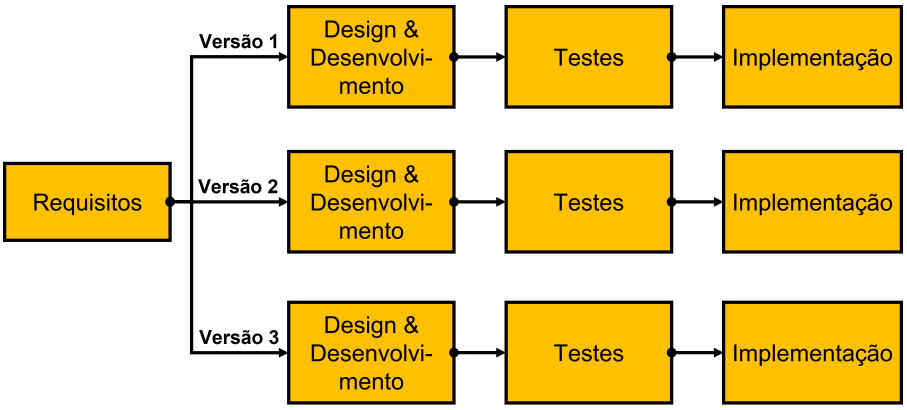
De acordo com Larman e Basili (2003), o modelo incremental é um modelo desenvolvido para superar os pontos fracos do modelo em cascata. Começa com um planejamento inicial e termina com a implementação com as interações intermediárias. A ideia básica por trás desse método é desenvolver um sistema por meio de ciclos repetidos (iterativos) e em porções menores de cada vez (incrementais), permitindo que os desenvolvedores de software aproveitem o que foi aprendido durante o desenvolvimento de partes ou versões anteriores do sistema.

Segundo Bhuvaneswari e Prabaharan (2013), suas vantagens são que ele produz um valor comercial já no início do ciclo de desenvolvimento, possui um melhor uso

de recursos que podem ser escassos por meio da definição de incremento adequada, pode acomodar algumas solicitações de alteração entre as versões, é um modelo mais focado no valor do cliente do que nas abordagens lineares e os problemas presentes podem ser detectados com antecedência. Já suas desvantagens são que requer uma documentação pesada, ele segue um conjunto definido de processos, define incrementos com base nas dependências de funções e recursos, requer mais envolvimento do cliente do que abordagens lineares, outro ponto seria que particionar as funções e recursos pode ser problemático e a integração entre iterações pode ser um problema se isso não for considerado durante o desenvolvimento.

Suas etapas são representadas na figura 12, onde cada versão nova incrementa a anterior, com correções, melhorias e possíveis alterações vindas dos requisitos.

Figura 12 – Fases do ciclo de desenovlvimento de software no modelo incremental.



Fonte: Adaptada de Bhuvaneswari e Prabaharan (2013).

3.3 MODELO EM V

É uma variante do modelo em cascata que enfatiza a verificação e validação do produto, os testes do produto são planejados em paralelo com a fase correspondente de desenvolvimento. Assim como o modelo em cascata, o ciclo de desenvolvimento em forma de V (FORSBERG; MOOZ, 1991) segue um caminho sequencial de execução de processos. Cada fase deve ser concluída antes do início da próxima e os testes são enfatizados neste modelo mais do que no modelo em cascata ou o incremental. Os procedimentos de teste são desenvolvidos no início do ciclo, antes que qualquer codificação seja feita, durante cada uma das fases que precedem a implementação (BUCANAC, 1999).

Os requisitos iniciam o modelo, como os modelos anteriores, porém, nesse caso, eles são enfatizados e isso traz maior garantia de sucesso ao longo do desenvolvimento, já que os testes são desenvolvidos com base nos requisitos. A fase de design de alto nível concentra-se na arquitetura e design do sistema. Um plano de teste de integração é criado nesta fase para testar a capacidade dos sistemas de software de funcionarem de maneira conjunta (BHUVANESWARI; PRABAHARAN, 2013).

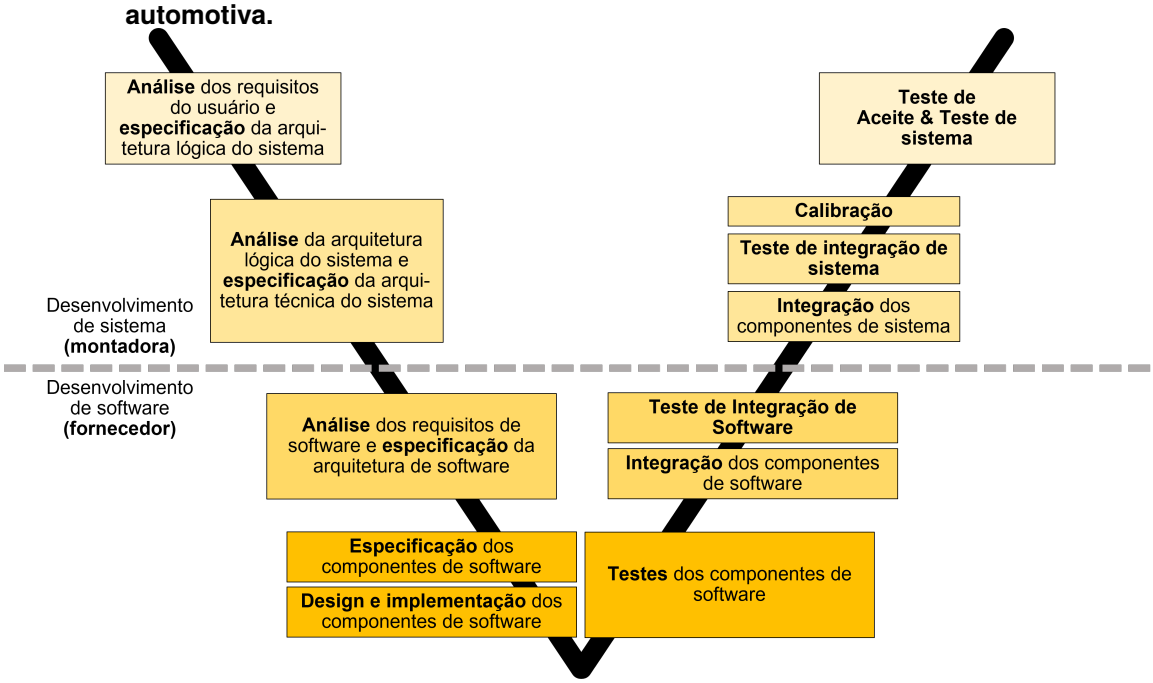
Como existem vários modelos de ciclo de desenvolvimento de software, cada um tem suas próprias vantagens e desvantagens. Por exemplo, se os requisitos são conhecidos de antemão, são bem compreendidos e necessita-se de controle total sobre o projeto em todos os momentos, pode-se usar o modelo em cascata. Contudo, o modelo em forma de V tem maior chance de sucesso em relação ao modelo em cascata devido ao desenvolvimento de planos de teste durante o ciclo de vida e o foco nos requisitos.

Quando se desenvolve um produto de software, existem diversos mercados que esses produtos podem atender e para cada tipo de mercado se aplica um método específico. Áreas como as indústrias automotiva, aeronáutica, ferroviária e médica, por exemplo, são áreas que fazem aplicação da denominada missão crítica para desenvolvimento de software. Isso significa que são áreas nas quais não se admite falha de software, já que uma falha pode levar à perda de vidas humanas. Portanto, a razão de se utilizar o modelo em V é que ele é fortemente baseado em requisitos e testes. Para essas indústrias, normalmente, quem desenvolve os requisitos são as empresas que estão contratando a empresa de desenvolvimento de software, para a área automotiva, os requisitos são desenvolvidos e definidos pelas montadoras de veículos.

Existem também os métodos ágeis de desenvolvimento de software, como por exemplo o Scrum (SCHWABER; BEEDLE, 2002) e o Desenvolvimento Ágil (HIGHSMITH, J. A.; HIGHSMITH, J., 2002). Porém, esses métodos não possuem um forte apelo ao desenvolvimento e rastreabilidade de requisitos, dessa forma, não serão abordados no presente trabalho. O modelo em V integra inspeção de qualidade e procedimentos de teste, diferenciando entre a visão de sistema e a visão de componentes de software. Portanto, é amplamente utilizado na indústria automotiva. A figura 13 mostra uma visão geral desse processo especificamente para a área automotiva.

Schauffele e Zurawka (2010) esclarece cada etapa do processo, descritas a seguir:

Figura 13 – Fases do ciclo de desenovlvimento de software no modelo em V na indústria automotiva.



Fonte: Adaptada de Schauffele e Zurawka (2010).

a) Análise dos requisitos do usuário e especificação da arquitetura lógica do sistema:

   O objetivo desta etapa do é definir a arquitetura lógica do sistema com base nos requisitos do usuário relevantes ao projeto. A arquitetura lógica do sistema inclui a definição da rede de funções, as interfaces de funções e a comunicação entre as funções em todo o veículo ou para um único subsistema. Esta etapa do processo ainda não produz nenhuma decisão no que diz respeito à implementação técnica.

b) Análise da arquitetura lógica do sistema e especificação da arquitetura técnica do sistema:

   A arquitetura lógica do sistema é a base para a especificação real da arquitetura técnica do sistema. A análise de alternativas técnicas de implementação é baseada em uma arquitetura lógica de sistema unificada e é embasada por uma variedade de metodologias de engenharia. A arquitetura técnica do sistema também inclui uma definição de todas as funções e subfunções que serão implementadas por meio do software. Essa definição também é chamada de requisitos de software.

c) Análise de requisitos de software e especificação de arquitetura de software:

Os requisitos de software definidos são analisados nessa etapa e a arquitetura do software é especificada. Ou seja, os limites e interfaces do sistema de software são definidos, com componentes de software, camadas de software, condições e modos de operação.

d) Especificação de componentes de software:
Esta etapa é seguida pela especificação dos componentes de software. O procedimento inicialmente assume um ambiente de “mundo ideal”. Isso significa que esta etapa ignora todos os detalhes de implementação.

e) Design, implementação e testes de componentes de software:
Na fase de design, os aspectos do “mundo real”, anteriormente ignorados, estão sujeitos a um exame minucioso. Neste ponto, todos os detalhes que afetam a implementação devem ser definidos. As decisões de design resultantes gerenciam a implementação dos componentes de software. No final desta etapa, os componentes de software são testados.

f) Integração de componentes de software e testes de integração de software:
Quando o desenvolvimento dos componentes de software é concluído e os componentes passaram nos testes unitários, a integração pode começar. Após a integração dos componentes em um sistema de software, os testes de integração concluem esta etapa.

g) Integração de componentes de sistema e testes de integração de sistema:
Nesta próxima etapa, o software deve ser instalado no hardware da ECU para fornecer à respectiva ECU recursos funcionais. As ECUs devem então ser integradas com os outros componentes do sistema eletrônico, como geradores de setpoint, sensores e atuadores. Em um teste de integração de sistema subsequente, a interação de todos os sistemas com a ECU é avaliada.

h) Calibração:
A calibração das funções do software da ECU compreende sua parametrização; a configuração dos valores dos parâmetros deve ser realizada individualmente para cada tipo ou variante de um determinado veículo. As configurações de parâmetros podem ser fornecidas pelo software na forma de valores característicos, curvas características e mapas característicos.

i) Teste de sistema e teste de aceite:

Finalmente, testes de sistema com foco na arquitetura lógica do sistema são executados, com os testes de aceite que se concentram nos requisitos do usuário final do produto.

3.3.1 Níveis de teste ao longo do ciclo de desenvolvimento em V

De acordo com ISTQB (2018b), os níveis de teste são definidos como grupos de atividades de teste que são organizados e gerenciados de maneira conjunta, onde cada nível é uma instância do processo de teste, consistindo em atividades executadas em relação ao software em um determinado nível de desenvolvimento, conforme a figura 13, desde as unidades individuais ou componentes de software, até os sistemas completos, como as ECUs, ou seja, os níveis de teste estão relacionados com as outras atividades dentro do ciclo de desenvolvimento de software do modelo em V.

Os níveis de teste são divididos entre testes unitários (de componentes), testes de integração, testes de sistema e testes de aceite. Os níveis de teste são caracterizados pela definição dos objetivos específicos (a base de testes, importante na derivação de casos de teste), objeto de teste (o que está sendo testado), defeitos e falhas típicas e, por fim, abordagens e responsabilidades específicas (ISTQB, 2018b). Para cada nível de teste, é necessário um ambiente de teste adequado, por exemplo, no teste de aceite é ideal que um ambiente de teste com características mais próximas do ambiente de produção esteja disponível, enquanto no teste de componentes (teste unitário) os desenvolvedores, geralmente, utilizam seu próprio ambiente de desenvolvimento para realização dos testes.

Os objetivos comuns entre todos os níveis de teste incluem reduzir os riscos, verificar os comportamentos funcional e não funcional de acordo com o que foi projetado e especificado nos requisitos, elaborar o nível de confiabilidade na qualidade do item testado, encontrar defeitos e evitar que os defeitos se espalhem para níveis mais altos de teste.

3.3.1.1 Testes unitários

O teste de componente, também definido como teste unitário ou modular, possui seu direcionamento em componentes de software que podem ser testados de maneira

individual. Usando a descrição do projeto em nível de componente como guia, orientações de controle são testadas para descobrir erros dentro dos limites do módulo. Se concentra na lógica de processamento interno e nas estruturas de dados dentro dos limites de um componente. Esse tipo de teste pode ser realizado em paralelo para vários componentes. O teste de componente pode cobrir funcionalidades, como a correção de cálculos, por exemplo, além de cobrir também características não funcionais, como busca de vazamentos de memória, por exemplo, e propriedades estruturais, como teste de decisão (PRESSMAN, 2010).

3.3.1.2 Testes de integração

O nível de testes de integração se concentra nas interações entre componentes ou vários sistemas e existem dois níveis diferentes de teste de integração descritos por ISTQB (2018b), que podem ser realizados em objetos de teste de tamanho variável. O primeiro é o teste de integração de componentes, que foca nas interações e interfaces entre componentes integrados. O teste de integração de componentes é executado após o teste unitário e geralmente é automatizado, esse nível de teste é realizado pelo fornecedor de software. Outro tipo de teste de integração é o teste de integração do sistema, que se concentra nas interações e interfaces entre sistemas e pacotes. O teste de integração do sistema pode ser feito após o teste do sistema ou em paralelo com as atividades de teste do sistema já em andamento.

3.3.1.3 Testes de sistema

Nesse nível de teste, o software é incorporado a outros elementos do sistema (hardware, pessoas, informações) e uma série de testes de integração e validação do sistema são realizados em seguida. Esses testes estão fora do escopo do processo de software e não são conduzidos apenas por engenheiros de software. No entanto, as etapas executadas durante o projeto e o teste do software podem aumentar muito a probabilidade de integração de software bem-sucedida no sistema maior (PRESSMAN, 2010).

O teste de sistema tem foco nas capacidades e no comportamento de todo um sistema, geralmente considerando as execuções das tarefas de ponta a ponta do

sistema e os comportamentos não funcionais que ocorrem ao executar tais tarefas. Para determinados sistemas, a verificação da qualidade dos dados utilizados nos testes também pode ser incluída como um objetivo nesse nível (MYERS; SANDLER; BADGETT, 2011). Assim como no teste de componente e no teste de integração, e em alguns casos os testes automatizados de regressão do sistema fornecem a confiabilidade de que as alterações não alteraram os recursos existentes ou a execução dos recursos de ponta a ponta do sistema.

O teste realizado neste nível geralmente produz informações que são usadas pelos gestores para tomar decisões de liberação do produto para produção. Este teste também pode satisfazer requisitos ou padrões legais e regulatórios, como os padrões ASPICE e ISO 26262. O ambiente para execução deste teste deve, preferencialmente, corresponder ao seu destino final do produto, ou ao ambiente de produção, já que suas características podem influenciar nos resultados dos testes, conforme mencionado por ISTQB (2018b).

3.3.1.4 Testes de aceite

O teste de aceite, da mesma forma como o teste do sistema, geralmente se concentra no comportamento e na capacidade de um sistema ou produto como um todo. Este teste que é conduzido pelo cliente em um esforço para exercer todos os recursos e funções necessários (PRESSMAN, 2010). Ele pode produzir informações para avaliar a situação do sistema para implantação e uso pelo cliente, que é o usuário final. Os defeitos podem ser encontrados durante a execução dos testes de aceite, mas encontrar defeitos muitas vezes não é um objetivo, e encontrar um número significativo de defeitos durante o teste de aceite pode, em alguns casos, ser considerado um grande risco de projeto, já que está num nível mais alto do ciclo de desenvolvimento. O teste de aceite também pode satisfazer requisitos ou padrões legais ou regulatórios específicos para este nível

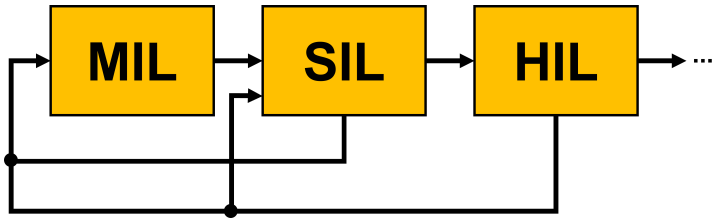
3.3.2 Testes baseados em modelos (MBT)

Para esta estratégia, os testes são elaborados baseados em algum modelo de algum aspecto necessário do produto, como uma função, um processo, uma estrutura

interna ou uma característica não funcional (por exemplo, confiabilidade). Podem ser realizados em três situações iniciais, são elas *Model-In-the-Loop* (MIL), *Software-In-the-Loop* (SIL) e *Hardware-In-the-Loop* (HIL) (JAIKAMAL, 2009).

O fluxo geral é linear, mas altamente iterativo, pois quaisquer alterações feitas devido aos resultados em estágios posteriores podem forçar o modelo de volta aos estágios anteriores para um novo teste, dependendo da extensão da alteração. A figura 14 ilustra o fluxo dos testes baseados em modelo (NIBERT; HERNITER; CHAMBERS, 2012).

Figura 14 – Fluxo de testes baseados em modelos.



Fonte: Adaptada de Nibert, Herniter e Chambers (2012).

Na etapa MIL do processo de testes, os modelos matemáticos do veículo, seus subsistemas e seu ambiente são conectados aos modelos funcionais de ECUs em um ambiente de integração que permite que tal sistema seja simulado e testado virtualmente (VULI; BADALAMENT; JAIKAMAL, 2010). Essa etapa acontece no nível de desenvolvimento do sistema do ciclo V (figura 13).

A etapa SIL conta com código e software básico para simular a ECU completa. Um objetivo da simulação SIL é mover sistematicamente cada novo modelo de controle um passo mais próximo do ambiente real da ECU. A maioria das empresas hoje usa alguma forma de geração automática de código para produzir código a partir dos modelos de controle. Portanto, faz sentido validar o código em relação ao modelo antes de integrá-lo a uma ECU. Isso dá aos controles e engenheiros de software um nível mais alto de confiança nas ferramentas e no processo específico em vigor para a geração de código. Outro objetivo do SIL é validar todo o software da ECU no PC antes que o hardware esteja disponível (VULI; BADALAMENT; JAIKAMAL, 2010). Essa etapa acontece no nível de desenvolvimento de software do ciclo V (figura 13).

Com uma abordagem diferente em relação às metodologias MIL e SIL que executam modelos e código virtualmente, o HIL se concentra na simulação de sensores e atuadores em tempo real e na execução do software de sistema de controle em

hardware de ECU real. A simulação HIL torna possível, por exemplo, averiguar se a potência de processamento da ECU é suficiente, se o código foi programado para ser executado nas taxas apropriadas e se a latência do sinal do sensor / atuador tem um impacto no desempenho do circuito fechado do sistema completo. Além disso, a simulação HIL torna possível simular e analisar os barramentos de comunicação física em uma rede única ou múltipla de ECU (VULI; BADALAMENT; JAIKAMAL, 2010). Essa etapa acontece a partir do nível de integração de software do ciclo V (figura 13).

3.4 REGULAMENTAÇÕES

Como já mencionado no presente trabalho, a inovação de produtos na indústria automotiva tem aumentado constantemente. Em 2019, cerca de 80% da inovação de produto ocorreu por meio do desenvolvimento de software. A diferenciação do produto por recursos eletrônicos explodiu o número de plataformas e variantes de veículos (**costa2016**). Cada variante é uma combinação única de recursos que terá diferentes interações e riscos que impactam a segurança. Esta situação exige a necessidade de definição, implementação e avaliação de processos adequados para o desenvolvimento do sistema e a coordenação de todas as partes interessadas (por exemplo, montadoras, fornecedores, empresas terceirizadas etc.) mais do que nunca (NEEMEH, 2020), para isso, existem alguns padrões principais que são trabalhados durante o desenvolvimento de software para que se possa garantir um maior nível de segurança e qualidade no produto final, com ênfase em testes em ECUs.

3.4.1 ASPICE

Automotive Software Performance Improvement and Capability dEtermination (ASPICE) é um padrão que fornece a estrutura para definir, implementar e avaliar o processo necessário para o desenvolvimento de sistemas de software na indústria automotiva. Essa estrutura pode ser estendida para incluir processos de outros domínios, como hardware e engenharia mecânica.

Na indústria automotiva, o padrão ASPICE vem tornando-se amplamente utilizado. Grandes montadoras de veículos, como Audi, BMW e Ford, avaliam seus fornecedores de eletrônicos e software com base na classificação da sua avaliação ASPICE

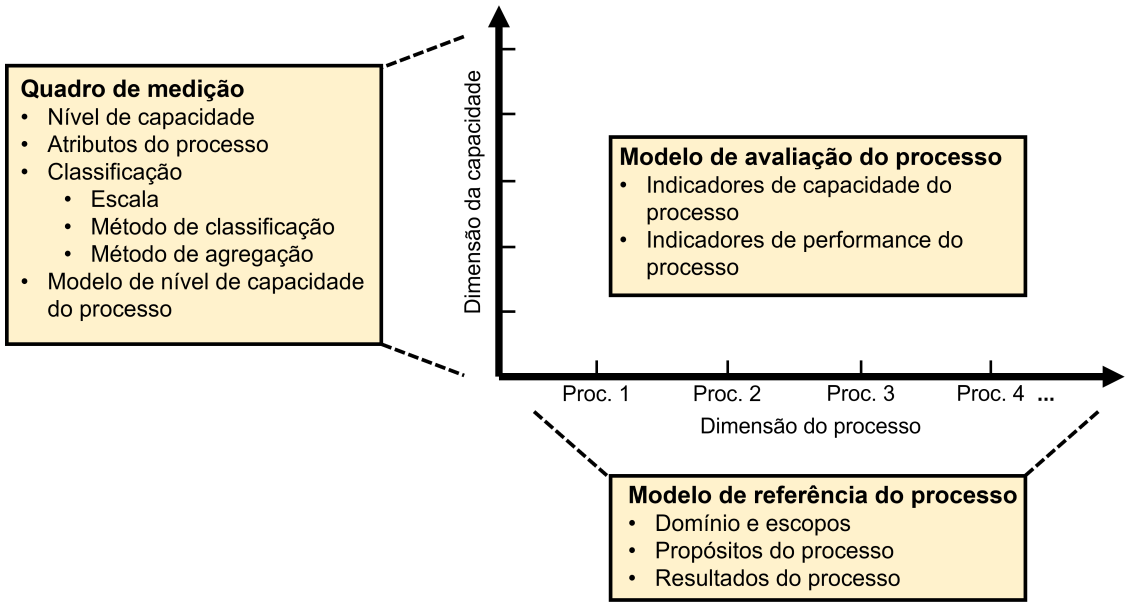
(NEEMEH, 2020). Ele fornece um processo de desenvolvimento mais controlado, garantindo a qualidade do produto, encurtando o cronograma de lançamento e reduzindo o impacto do custo no desenvolvimento do produto devido a problemas de qualidade identificados em estágios posteriores do desenvolvimento do produto. As montadoras podem utilizar a estrutura ASPICE para avaliar a capacidade de qualidade do processo de seu fornecedor durante a seleção do fornecedor, além de poderem definir seu próprio processo de desenvolvimento de sistema para ser compatível com o padrão ASPICE, o que ajudará a avaliar e melhorar a capacidade do processo como um todo (ASPICE, 2017).

O ASPICE possui seu próprio Modelo de Referência de Processos (PRM - *Process Reference Model*) que é customizado considerando as necessidades específicas da indústria automotiva. O Modelo de Avaliação de Processo ASPICE (PAM - *Process Assessment Model*) utiliza o PRM ao realizar uma avaliação. Nesse padrão, a determinação da capacidade é baseada em uma estrutura bidimensional: Dimensão do Processo pela Dimensão da Capacidade, conforme a figura 15. O modelo de avaliação de processo seleciona processos de um modelo de referência de processo e complementa com indicadores. Esses indicadores suportam a coleta de evidências objetivas que permitem a um avaliador atribuir classificações para processos de acordo com a dimensão da capacidade. A dimensão do processo define o PRM em áreas do processo e seu escopo, propósitos e resultados. A dimensão de capacidade consiste nos níveis de capacidade e atributos de processo para as áreas identificadas no PRM (NEEMEH, 2020).

Os processos são agrupados por categoria e, em um segundo nível, em grupos de processos de acordo com o tipo de atividade que eles abordam. Existem 3 categorias de processos: Processos primários do ciclo de vida, Processos organizacionais do ciclo de vida e Processos de apoio do ciclo de vida. Cada processo é descrito em termos de uma declaração de propósito. A declaração de propósito contém os objetivos funcionais exclusivos do processo quando executado em um ambiente específico. Para cada declaração de propósito, uma lista de resultados específicos é associada, como uma lista de resultados positivos esperados do desempenho do processo. Para a dimensão do processo, o modelo de referência do processo ASPICE fornece o conjunto de processos mostrado na figura 16 (ASPICE, 2017).

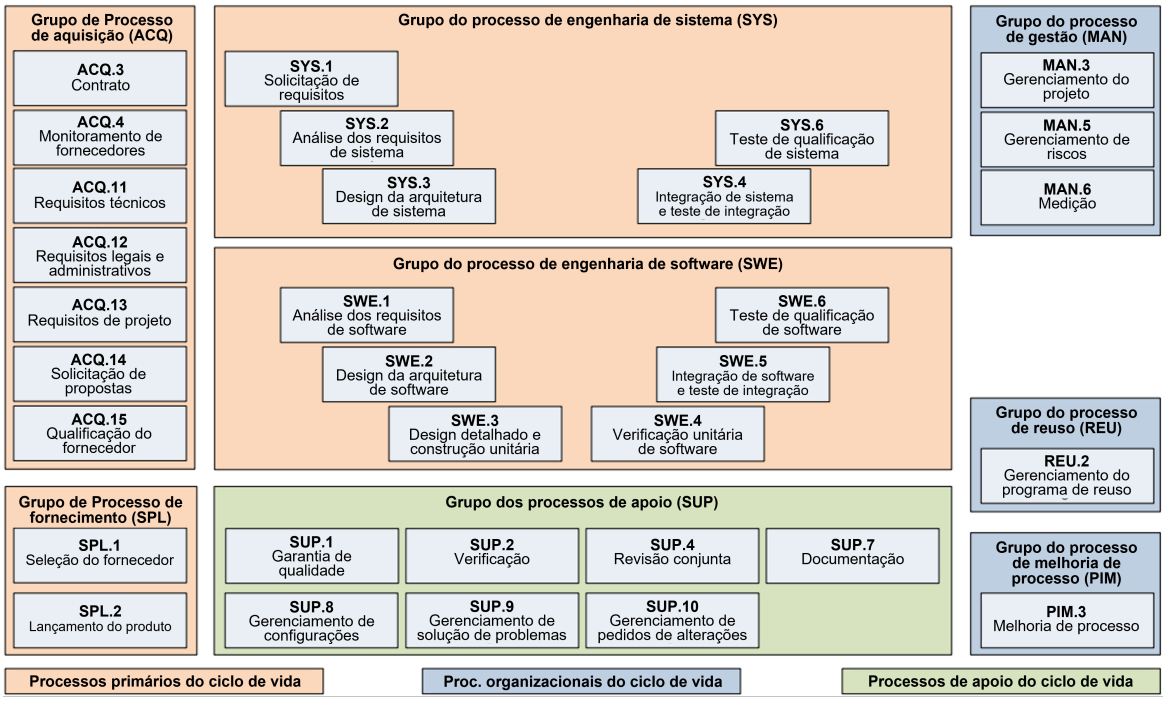
A dimensão de capacidade consiste em níveis de capacidade (CL - *Capability*

Figura 15 – Estrutura para determinação da capacidade pelo ASPICE.



Fonte: Adaptada de ASPICE (2017).

Figura 16 – Conjuntos de processos definidos pelo ASPICE.



Fonte: Adaptada de ASPICE (2017).

Quadro 1 – Níveis de capacidade ASPICE.

Nível 0	Processo incompleto.
	O processo não é implementado ou não atinge seu propósito.
Nível 1	Processo realizado.
	O processo implementado atinge seu propósito.
Nível 2	Processo gerenciado.
	O processo executado anteriormente descrito agora é implementado de forma gerenciada (planejada, monitorada e ajustada) e seus produtos de trabalho são devidamente estabelecidos, controlados e mantidos.
Nível 3	Processo estabelecido.
	O processo gerenciado descrito anteriormente é agora implementado usando um processo definido que é capaz de atingir os resultados do processo.
Nível 4	Processo previsível.
	O processo estabelecido anteriormente descrito agora opera de forma preditiva dentro de limites definidos para atingir os resultados do processo. As necessidades de gerenciamento quantitativo são identificadas, os dados de medição são coletados e analisados para identificar as causas atribuíveis de variação. Ações corretivas são tomadas para abordar as causas atribuíveis de variação.
Nível 5	Processo inovador.
	O processo previsível descrito anteriormente é agora continuamente aprimorado para responder às mudanças organizacionais.

Fonte: Adaptado de ASPICE (2017).

Levels) que são subdivididos em atributos de processo (PA - *Process Attributes*) (ASPICE, 2017). Os atributos do processo são características de um processo que podem ser avaliados em uma escala de realização, fornecendo uma medida da capacidade do processo e eles são aplicáveis a todos os processos. Um nível de capacidade é um conjunto de atributos de processo que trabalham juntos para fornecer um aprimoramento importante na capacidade de executar um processo. Cada atributo aborda um aspecto específico do nível de capacidade. Os níveis constituem uma forma racional de progredir através da melhoria da capacidade de qualquer processo e estão demonstrados no quadro 1 (ISTQB, 2018a).

O ASPICE define dois tipos de indicadores: práticas de base (BP - *Base Practices*) e produtos de trabalho (WP - *Work Products*). Além disso, as práticas genéricas (GP - *General practices*) e os recursos são definidos. A avaliação dos atributos do processo é baseada no nível de implementação dos indicadores em quatro níveis de classificação (ASPICE, 2017):

- N (*None*): não cumprido (0% até $\leq$ 15%)
- P (*Partly*): parcialmente cumprido ($>$ 15% até $\leq$ 50%)
- L (*Largely*): amplamente cumprido ($>$ 50% até $\leq$ 85%)

- F (*Fully*): totalmente cumprido (> 85% até ≤ 100%)

Para que um processo atinja um determinado nível de capacidade, os indicadores do nível de capacidade a ser alcançado devem estar amplamente cumpridos (L). Os indicadores dos níveis de capacidade mais baixos devem ser totalmente cumpridos (F) (ISTQB, 2018a).

Em suma, o ASPICE se trata de um conjunto de ações que avaliam a maturidade de uma empresa de desenvolvimento, levando em consideração seus processos, fazendo um alinhamento com todas as etapas do ciclo de desenvolvimento de software em forma de V, sendo fator importante se tratando de concorrência entre fornecedores para montadoras.

3.4.2 ISO 26262 (*FUNCTIONAL SAFETY*)

A ISO 26262 se refere à segurança funcional, tendo como objetivo diminuir o risco de ocorrência de uma falha em algum subsistema do veículo, a qual pode vir a comprometer a vida das pessoas no veículo (ISO, 2018).

Para que se diminua o risco da ocorrência de falha, a ISO 26262 aborda duas frentes. A primeira foca em diminuir as falhas aleatórias, que são falhas que podem ocorrer em componentes de hardware, por exemplo, criando-se redundâncias e sistemas de diagnóstico para que seja possível fazer o monitoramento das possíveis falhas de componentes de hardware. Na ISO 26262 existem requisitos que determinam como reduzir a ocorrência dessas falhas aleatórias, além de processos e métodos que devem ser aplicados durante todo o ciclo de vida do produto (concepção, desenvolvimento, operação e descomissionamento) no intuito de mitigar as falhas sistêmicas. A segunda abordagem busca reduzir as falhas sistêmicas, que são falhas introduzidas durante o desenvolvimento de produto, que estão relacionadas ao processo de desenvolvimento de software (ISO, 2018), seja no design ou codificação.

Na ISO 26262 existe uma classificação de criticidade das funções da ECU e componentes de software, essa classificação é denominada ASIL, sendo determinada já no início do processo de desenvolvimento, na realização da análise de riscos. Sua determinação é baseada em letras que vão de A até D, sendo um componente classificado como D, a maior letra, o componente em questão apresenta uma criticidade suscetível à segurança, portanto, é necessário ter um alto rigor no desenvolvimento da

função deste componente de software, ao passo que um componente classificado com um nível de criticidade mais baixo não exige o mesmo nível de rigor. Desta forma, de acordo com Caminski (2018), o ASIL fornece orientações para que se possa escolher os métodos adequados para alcançar certo nível de integridade do produto, sendo que tais orientações são destinadas a complementar as práticas de segurança no desenvolvimento. Assim, os automóveis atuais são fabricados em um nível de segurança alto e a ISO 26262 é feita para padronizar algumas práticas ao longo de toda indústria.

A norma ISO 26262 identifica os requisitos mínimos de teste dependendo do ASIL do componente de software ou da função da ECU em questão, relacionando as técnicas de acordo com os níveis de testes, essa relação abrange as técnicas de caixa-preta e caixa-branca (testes de componentes e de integração). Estão representados nos quadros 2, 3 e 4, onde "+" significa que é recomendado e "++" significa que é fortemente recomendado (mandatório), isso para cada nível ASIL (A, B, C ou D) (ISO, 2018).

Quadro 2 – Métodos de testes unitários.

Métodos de testes unitários		ASIL			
		A	**B**	**C**	**D**
1	Teste de declaração e cobertura	++	++	+	+
2	Teste de decisão e cobertura	+	++	++	++
3	Valor da declaração e do teste de decisão	+	+	+	++

Fonte: Adaptado de ISO (2018).

Quadro 3 – Métodos de testes de integração.

Métodos de testes de integração		ASIL			
		A	**B**	**C**	**D**
1	Análise dos requisitos	++	++	++	++
2	Geração e análise de classes de equivalência (particionamento)	+	++	++	++
3	Análise do valor limite	+	++	++	++
4	Suposição de error baseada em experiência	+	+	+	+
5	Teste de cobertura de funções	+	+	++	++
6	Teste de cobertura total	+	+	++	++

Fonte: Adaptado de ISO (2018).

Com isso, pode-se observar a importância do conhecimento das técnicas de testes tanto para testes funcionais quanto para caixa-branca, já que garantem o bom funcionamento do veículo, assegurando a segurança funcional e prezando pela vida dos consumidores de veículos.

Quadro 4 – Métodos de testes de sistema/aceite.

Métodos de testes de sistema/aceite		ASIL			
		A	B	C	D
1	Análise dos requisitos	++	++	++	++
2	Geração e análise de classes de equivalência (particionamento)	+	++	++	++
3	Análise do valor limite	+	+	++	++
4	Suposição de error baseada em experiência	+	+	++	++
5	Análise de dependência funcional	+	+	++	++
6	Análise de casos de uso operacionais	+	++	++	++

Fonte: Adaptado de ISO (2018).

3.4.3 ISTQB

A ISTQB (*International Software Testing Qualifications Board*) (ISTQB, 2020) é uma organização que concentra estudos no aprimoramento dos processos de testes de software. Ela certifica profissionais responsáveis pelos testes e determina os passos que o testador deve seguir para que possa obter melhores resultados e testes com maior qualidade.

Para os dois padrões supracitados, que são aplicados a testes funcionais em ECUs, o ASPICE e a ISO 26262, a ISTQB determina que a pessoa responsável por realizar os testes siga as seguintes diretrizes. Para o ASPICE, o testador deve ser capaz de:

- saber as duas dimensões do ASPICE;
- explicar os níveis de capacidade 0 a 3 do ASPICE;
- explicar o significado dos 4 níveis de classificação e os indicadores de capacidade do ASPICE da perspectiva do teste;
- explicar os requisitos do ASPICE para a estratégia de teste, incluindo a estratégia de teste de regressão;
- saber os requisitos do ASPICE para a documentação do teste;
- projetar uma estratégia de verificação (em contraste com uma estratégia de teste) e critérios para verificação da unidade; e
- explicar os diferentes requisitos de rastreabilidade do ASPICE da perspectiva do teste.

Para a ISO 26262, o testador deve ser capaz de:

- explicar o objetivo de segurança funcional para sistemas E / E;
- saber sua contribuição como testador para a cultura de segurança;

- apresentar o papel do testador na estrutura do ciclo de vida de segurança conforme ISO 26262;
- saber os volumes (títulos das partes) da ISO 26262 que são relevantes para ele;
- saber os níveis de criticidade do ASIL;
- explicar a influência do ASIL nas técnicas de design de teste aplicáveis e tipos de teste para testes estáticos e dinâmicos, além da extensão do teste resultante; e
- aplicar a tabela de métodos selecionados da ISO 26262.

Dessa maneira, os profissionais responsáveis pelos testes estarão aptos a realizar a verificação destes padrões de maneira correta e eficaz, garantindo maior segurança e qualidade ao processo.

4 TESTES EM ECUS

Como visto no desenvolvimento do presente trabalho, tipicamente, o ciclo de vida do produto automotivo segue o modelo V, o qual possui um rigor muito forte em testes e validações. O lado direito do V é constituído de vários níveis de testes com diferentes objetivos, de modo a reduzir o risco de falha da ECU durante sua operação no veículo, garantir a qualidade dos componentes e atender aos requisitos e padrões específicos da indústria automotiva. Estes diferentes níveis de testes envolvem especialistas em todos os estágios do projeto, seja no fornecedor ou na montadora.

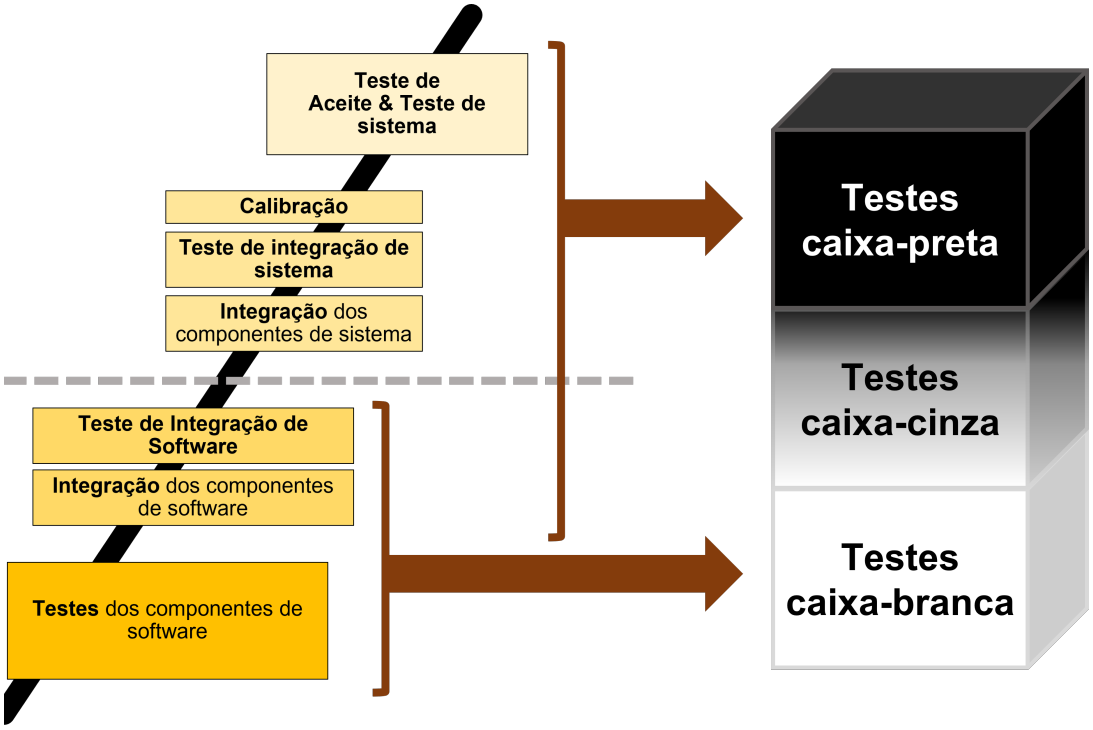
Grupos de atividades de teste destinados a testar características específicas de um sistema de software, ou parte de um sistema, com base em objetivos específicos de requisito são denominados tipos de testes. Tais objetivos podem incluir a avaliação das características de qualidade funcional (tais como integridade, correção e adequação), avaliação das características de qualidade não funcionais (como confiabilidade, eficiência de performance, segurança, compatibilidade e usabilidade), avaliação da estrutura ou arquitetura do componente ou sistema para verificar se está correta, completa e especificada, avaliação dos efeitos das alterações, como a confirmação da correção dos defeitos (teste de confirmação) e procurar alterações não intencionais no comportamento como resultado de alterações no software ou no ambiente (teste de regressão) (BLACK, 2009).

Os conjuntos de testes podem ser divididos em duas categorias principais, levando em consideração o escopo do teste, são elas os chamados testes Caixa-branca e os testes Funcionais (Caixa-preta), conforme representado na figura 17. Existe ainda uma categoria denominada testes Caixa-cinza, que nada mais é do que uma combinação das metodologias utilizadas entre as duas outras categorias citadas. Essa última não é mandatória, mas sua utilização implica na criação de testes mais criteriosos, já que mescla as metodologias (SILVA, 2020).

Antes de tratarmos das técnicas especificamente, é importante levar em consideração alguns princípios a serem considerados na realização de testes de software, sendo que muitos desses princípios podem parecer óbvios, mas são frequentemente esquecidos em situações reais. Esses princípios foram citados por Myers, Sandler e Badgett (2011), são eles:

- Definição da saída ou resultado esperado.

Figura 17 – Tipos de técnicas de testes associados aos níveis de testes.



Fonte: Autoria própria.

- Um desenvolvedor deve evitar tentar testar seu próprio software.
- Qualquer processo de teste deve incluir uma inspeção completa dos resultados de cada teste.
- Os casos de teste devem ser escritos para condições de entrada que são inválidas e inesperadas, bem como para aquelas que são válidas e esperadas.
- Examinar um software para ver se ele não faz o que deveria fazer é apenas metade da batalha; a outra metade é ver se o programa faz o que não deve fazer.
- Evitar descartar casos de teste, a menos que o software seja realmente algo descartável.
- Não planejar um esforço de teste sob a suposição errônea de que nenhum erro será encontrado.
- A probabilidade de haver mais erros em uma seção de um software é proporcional ao número de erros já encontrados naquela seção.
- O teste é uma tarefa extremamente criativa e intelectualmente desafiadora.

As técnicas de teste possuem o objetivo de ajudar a identificar condições, casos e dados de teste. O uso dessas técnicas nas atividades de análise, desenvolvimento e implementação do teste pode variar desde muito informal, onde há pouca ou ne-

nhuma documentação, até muito formal, onde a documentação é mais completa e possui rastreabilidade e fácil acesso para consulta (LUO, 2001). O nível adequado da formalidade depende do contexto dos testes, incluindo a maturidade dos processos de teste e de desenvolvimento, das restrições de tempo, dos requisitos normativos ou de segurança, do conhecimento e das habilidades das pessoas envolvidas e do modelo de desenvolvimento de software que está sendo seguido, modelo em V (FORSBERG; MOOZ, 1991) no caso da indústria automotiva.

4.1 TÉCNICAS DE TESTES FUNCIONAIS

Como apresentado por Hooda e Chhillar (2015), o teste funcional de um sistema envolve testes que avaliam as funções que o sistema deve executar. Os requisitos funcionais podem ser descritos nos produtos de trabalho, tais como especificações de requisitos de negócios, históricos de usuários, casos de uso ou especificações funcionais, ou podem até mesmo não serem documentados. As funções de um software são “o quê” o sistema deve fazer, e é isso que é verificado nos testes funcionais. Esses testes devem ser realizados em todos os níveis de teste (por exemplo, os testes de componentes podem ser baseados em uma especificação de componente), embora o foco seja diferente em cada nível, conforme supracitado nos itens anteriores. A eficácia do teste funcional pode ser medida por meio da cobertura funcional, que pode ser definida como uma extensão em que alguma funcionalidade foi exercida por testes e é expressa como uma porcentagem do(s) tipo(s) de elemento que está sendo coberto. Por exemplo, usando rastreabilidade entre testes e requisitos funcionais, a porcentagem desses requisitos que são tratados por teste pode ser calculada, potencialmente identificando lacunas de cobertura. Eles não requerem nenhum conhecimento sobre a estrutura interna do sistema de software e é também nesta categoria que os testes de segurança funcional estão incluídos (MONITORA, 2019). Seu objetivo é determinar a segurança de um produto de software e exigir um procedimento sistemático, planejado, executado e documentado.

As técnicas funcionais podem ser classificadas entre caixa-preta ou baseada na experiência. As técnicas de teste caixa-preta se concentram nas entradas e saídas do objeto de teste sem referência a sua estrutura interna, ou seja, essa técnica verifica se as funções do software estão funcionando de acordo com os requisitos, mas não

verifica as linhas de código fonte da ECU (JOVANOVIĆ, 2006).

As metodologias dessa técnica possuem algumas características em comum de acordo com ISTQB (2018b), são elas: as condições, os casos e os dados de teste são derivados de uma base de teste que são provenientes de requisitos de software, especificações, casos de uso e históricos de usuários; os casos de teste podem ser usados para detectar lacunas entre os requisitos e as suas implementações, bem como divergências nos requisitos; e a cobertura é medida com base nos itens testados na base de teste e na técnica aplicada nesta base.

Já as técnicas de teste com base na experiência, se baseiam no conhecimento dos desenvolvedores, dos testadores e dos usuários para elaborar e aplicar os testes (ISTQB, 2018b).

4.1.1 Técnicas de teste caixa-preta

As técnicas de tabela de decisão, partição de equivalência e análise do valor limite, detalhadas adiante, são aplicadas em sistemas puramente combinatórios, ou seja, não fazem uso do efeito memória, onde uma saída depende apenas de uma combinação dos valores das entradas. Técnicas de teste combinatórias são úteis para testar a implementação de requisitos do sistema que especificam como diferentes condições de combinações levam a resultados diferentes.

4.1.1.1 Teste de tabela de decisão

As tabelas de decisão são consideradas uma boa maneira de se registrar regras de requisitos complexas que um sistema deve implementar. É uma técnica que permite testar a implementação de requisitos que especificam diferentes condições de combinação e como essas condições levam a diferentes resultados. Essa técnica já é bastante conhecida, principalmente pela sua similaridade a técnica Tabela-Verdade, utilizada em sistemas digitais puramente combinatórios (SILVA, 2020).

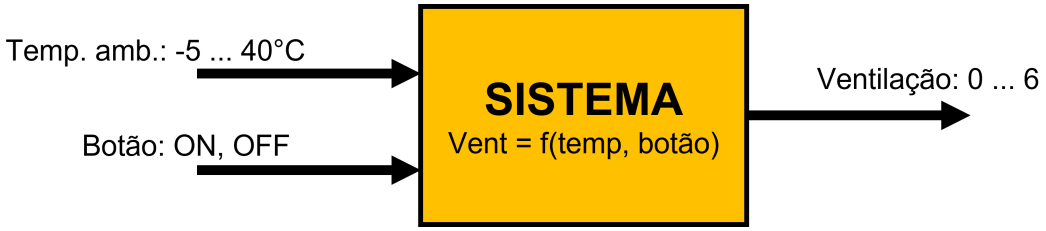
Durante a criação das tabelas, o testador deve identificar as entradas (condições) e quais são as saídas (ações resultantes da combinação das entradas) do sistema. Diferente do que acontece na técnica Tabela-Verdade de sistemas digitais, além de se usar valores booleanos, podem ser utilizados também valores discretos,

números inteiros ou até mesmo intervalos numéricos (ISTQB, 2018b).

Para essa técnica, o teste é projeto a fim de exercitar todas as condições relevantes de condições, excluindo-se as que são impossíveis de ocorrer ou que não são relevantes em uma ação resultante. Uma tabela de decisão completa possui colunas suficientes para cobrir todas as combinações de condições necessárias para o teste. Excluindo colunas que não afetam o resultado, o número de casos de teste pode diminuir consideravelmente. O padrão de cobertura mínima comum para o teste da tabela de decisão é ter pelo menos um caso de teste por regra de decisão na tabela. Isso normalmente envolve cobrir todas as combinações de condições. A vantagem do teste da tabela de decisão é que ele ajuda a identificar todas as combinações importantes de condições, algumas das quais, de outra forma, poderiam ser negligenciadas. Também ajuda a encontrar quaisquer lacunas na especificação de requisitos, já que os testes são provenientes deles. Pode ser aplicada a todas as situações em que o comportamento do software depende de uma combinação de condições, em qualquer nível de teste (PRESSMAN, 2010).

A seguir, tem-se um exemplo na figura 18 de um caso para teste de um sistema de ventilação, onde o sistema opera em função de duas entradas, a temperatura ambiente, que pode variar de -5°C até 40°C, e um botão de on/off, que possui dois estados, ligado (ON) e desligado (OFF), e , por fim, a saída é a intensidade da ventilação que possui múltiplos níveis de acionamento de 0 até 6. O sistema (ECU) controla o nível de ventilação com base nas duas variáveis de entrada (temp. e botão). Criou-se então a tabela 5 onde se relacionam todas as condições de entrada com as condições de saída esperadas, de modo que cada linha da tabela se refere a um caso de teste diferente (CT significa caso de teste).

Figura 18 – Exemplo de um caso de teste.



Fonte: Autoria própria.

Como é possível observar nesse exemplo, a quantidade de vetores de testes, que são as linhas da tabela, é consideravelmente alta mesmo para um exemplo consi-

Quadro 5 – Exemplo de condições da técnica de Tabela de decisão.

CT	Entradas		Saída
	Botão	Temp. amb. [°C]	Vent.
1	OFF	-5	0
2	OFF	-4	0
3	OFF	-3	0
4	OFF	-2	0
5	OFF	-1	0
6	OFF	0	0
7	OFF	1	0
8	OFF	2	0
9	OFF	3	0
10	OFF	4	0
11	OFF	5	0
12	OFF	6	0
13	OFF	7	0
14	OFF	8	0
15	OFF	9	0
16	OFF	10	0
17	OFF	11	0
18	OFF	12	0
19	OFF	13	0
20	OFF	14	0
21	OFF	15	0
22	OFF	16	0
23	OFF	17	0
24	OFF	18	0
25	OFF	19	0
26	OFF	20	0
27	OFF	21	0
28	OFF	22	0
29	OFF	23	0
30	OFF	24	0
31	OFF	25	0
32	OFF	26	0
33	OFF	27	0
34	OFF	28	0
35	OFF	29	0
36	OFF	30	0
37	OFF	31	0
38	OFF	32	0
39	OFF	33	0
40	OFF	34	0
41	OFF	35	0
42	OFF	36	0
43	OFF	37	0
44	OFF	38	0
45	OFF	39	0
46	OFF	40	0

CT	Entradas		Saída
	Botão	Temp. amb. [°C]	Vent.
47	ON	-5	0
48	ON	-4	0
49	ON	-3	0
50	ON	-2	0
51	ON	-1	0
52	ON	0	0
53	ON	1	0
54	ON	2	0
55	ON	3	0
56	ON	4	0
57	ON	5	0
58	ON	6	0
59	ON	7	0
60	ON	8	0
61	ON	9	0
62	ON	10	0
63	ON	11	0
64	ON	12	0
65	ON	13	0
66	ON	14	0
67	ON	15	0
68	ON	16	0
69	ON	17	0
70	ON	18	0
71	ON	19	0
72	ON	20	0
73	ON	21	0
74	ON	22	0
75	ON	23	0
76	ON	24	1
77	ON	25	1
78	ON	26	2
79	ON	27	2
80	ON	28	3
81	ON	29	3
82	ON	30	4
83	ON	31	4
84	ON	32	5
85	ON	33	5
86	ON	34	6
87	ON	35	6
88	ON	36	6
89	ON	37	6
90	ON	38	6
91	ON	39	6
92	ON	40	6

Fonte: Autoria própria (2021).

derado simples, possuindo apenas duas variáveis de entrada. Para um sistema mais complexo, um sensor de estacionamento de um veículo, por exemplo, essa técnica torna-se inviável, já que exigiria uma quantidade de testes impraticável, principalmente pela questão de horas de engenharia, já que em um projeto automotivo, geralmente os prazos são justos e devem ser respeitados rigorosamente, correndo risco de ser agregado um custo extra ao projeto. Portanto, a técnica a seguir seria mais adequada.

4.1.1.2 Particionamento de equivalência

De acordo com Pressman (2010), o particionamento de equivalência é um método de teste de caixa preta que divide o domínio de entrada de um sistema em classes de dados das quais os casos de teste podem ser derivados. Nessa técnica, um caso de teste ideal descobre uma classe de erros que podem exigir que muitos casos de teste sejam executados antes que o erro geral seja observado, como a tabela de decisão. O design de caso de teste para particionamento de equivalência é baseado em uma avaliação de classes de equivalência para uma condição de entrada. Uma classe de equivalência representa um conjunto de estados válidos ou inválidos para as condições de entrada.

Normalmente, uma condição de entrada é um valor numérico específico, um intervalo de valores, um conjunto de valores relacionados ou uma condição booleana. As classes de equivalência podem ser definidas de acordo com as seguintes diretrizes, segundo Pressman (2010):

1. Se uma condição de entrada especifica um intervalo, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas.
2. Se uma condição de entrada requer um valor específico, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas.
3. Se uma condição de entrada especifica um membro de um conjunto, uma classe de equivalência válida e uma inválida são definidas.
4. Se uma condição de entrada for booleana, uma classe válida e uma classe inválida são definidas.

O particionamento de equivalência divide os dados em partições, ou classes de equivalência, de tal maneira que todos os membros de uma determinada classe deve ser processado da mesma forma (KANER; PADMANABHAN; HOFFMAN, 2013),

(JORGENSEN, 2018).

Existem partições de equivalência para valores válidos e inválidos, de acordo com ISTQB (2018b), são os seguintes.

- Valores válidos são valores que devem ser aceitos pelo componente ou sistema. Uma partição de equivalência contendo este tipo de valores é chamada de "partição de equivalência válida".
- Valores inválidos são valores que devem ser rejeitados pelo componente ou sistema. Uma partição de equivalência contendo estes valores é chamada de "partição de equivalência inválida".
- As partições podem ser identificadas para qualquer elemento de dados relacionado ao objeto de teste, incluindo entradas, saídas, valores internos, valores relacionados a tempo (por exemplo, antes ou depois de um evento) e para parâmetros de interface (por exemplo, componentes integrados sendo testados durante o teste de integração da ECU).
- Se necessário, qualquer partição pode ser dividida em subpartições.
- Cada valor deve pertencer a uma e apenas uma partição de equivalência.
- Quando partições de equivalência inválidas são usadas em casos de teste, elas devem ser testadas individualmente, ou seja, não podem ser combinadas com outras partições de equivalência inválidas, para garantir que as falhas não sejam mascaradas. Falhas podem ser mascaradas quando várias falhas ocorrem ao mesmo tempo, mas apenas uma é visível, fazendo com que as outras falhas não sejam detectadas.

Em ISTQB (2018b), comenta-se que para obter se uma cobertura de completa utilizando essa técnica, os casos de teste devem cobrir todas as partições identificadas (incluindo partições inválidas) usando no mínimo um valor de cada partição. Essa técnica é aplicável em todos os níveis de teste.

Seguindo o mesmo exemplo da técnica anterior (figura 18), um sistema de ventilação com duas entradas (temperatura ambiente e botão on/off) e uma saída (nível de ventilação), a quantidade de testes diminui muito, quando aplicada a técnica de particionamento de equivalência.

Aplicando a técnica de particionamento de equivalência para esse exemplo, criou-se partições (classes) para cada variável de entrada. No caso do botão on/off, por ser uma entrada binária, tem-se duas classes, conforme o quadro 6, classe Ligado e

Quadro 6 – Exemplo de condições da técnica de particionamento de equiva - Classes do botão.

Ligado	Desligado
ON	OFF

Fonte: Autoria própria (2021).

Quadro 7 – Exemplo de condições da técnica de particionamento de equiva - Classes da temperatura.

Inválida 1	Válida 0	Válida 1	Válida 2	Válida 3	Válida 4	Válida 5	Válida 6	Inválida 2
<-5°C	-5°C,..., 23°C	24°C, 25°C	26°C, 27°C	28°C, 29°C	30°C, 31°C	32°C, 33°C	34°C, ..., 40°C	>40°C

Fonte: Autoria própria (2021).

classe Desligado. Para a faixa de temperatura (quadro 7), considerando que essa faixa opera de maneira diferente para os níveis de ventilação, tem-se então oito partições (classes).

Considerando o cenário deste exemplo, o mínimo de casos de teste que precisam ser executados é de 18, fazendo-se a combinação das classes do botão com as classes dos valores de temperatura ambiente. É notória a redução da quantidade de casos para os testes em relação à técnica de tabela de decisão (96 casos de teste), mesmo abrangendo-se todas as condições, portanto, se torna uma técnica mais vantajosa, quando o objetivo é otimizar o tempo e os recursos empregados nos testes necessários no desenvolvimento do projeto.

4.1.1.3 Análise de valor limite

De acordo com Pressman (2010), um número maior de erros ocorre nos limites do domínio de entrada, em vez de no "centro". É por esta razão que a análise de valor limite foi desenvolvida como uma técnica de teste. A análise de valor limite leva a uma seleção de casos de teste que exercitam os valores limite.

Myers, Badgett et al. (2004) apresentam a análise de valor limite como uma técnica de design de caso de teste que complementa o particionamento de equivalência. Em vez de selecionar qualquer elemento de uma classe de equivalência, a análise de valor limite leva à seleção de casos de teste nas “bordas” da classe. Em vez de focar apenas nas condições de entrada, a análise de valor limite também deriva casos de teste do domínio de saída. As diretrizes para a análise de valor limite são semelhantes em muitos aspectos às fornecidas para partição de equivalência. A maioria dos engenheiros

de software executa análise de valor limite intuitivamente em algum nível. Ao aplicar essas diretrizes, o teste de limite será mais completo, com maior probabilidade de detecção de erros (BEIZER, 2003).

Segundo ISTQB (2018b), é importante lembrar que os limites especificados e implementados podem ser deslocados para posições acima ou abaixo de suas posições pretendidas, podem ser omitidos por completo ou podem ser complementados com limites adicionais indesejados. A análise e o teste do valor-limite revelarão quase todos esses defeitos forçando o software a mostrar comportamentos de uma partição diferente daquela à qual o valor limite deveria pertencer. Essa metodologia pode ser aplicada em quaisquer níveis de testes, porém é mais utilizada quando há a necessidade de testar requisitos que precisam e um intervalo numérico (JORGENSEN, 2018).

Seguindo no exemplo citado nas técnicas anteriores (figura 18), como a análise do valor limite é uma extensão do particionamento de equivalência, aproveitou-se as classes definidas na tabela 7 para então analisar os valores limites de cada classe, conforme as definições estabelecidas por Beizer (2003) e Jorgensen (2018). Portanto para a classe “Válida 0”, os valores de teste para a variável Temperatura ambiente devem ser -6°C, -5°C, -4°C, 22°C, 23°C e 24, para a classe “Válida 6”, os valores testados devem ser 33°C, 34°C, 35°C, 39°C, 40°C e 41°C. Para as classes “Válida 1”, “Válida 2”, “Válida 3”, “Válida 4” e “Válida 5”, esses são os vetores de testes que garantem uma maior probabilidade de detecção de falhas.

4.1.1.4 Teste de transição de estado

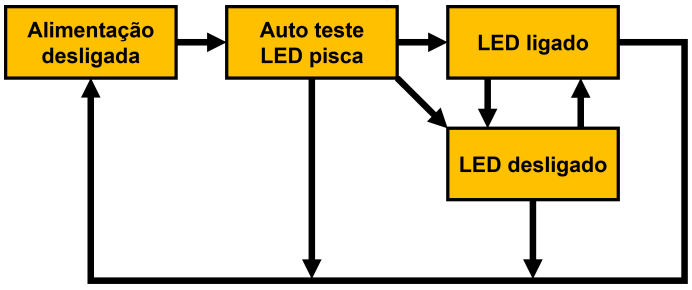
As técnicas vistas anteriormente são apropriadas para sistemas combinatórios, já quando se trata de sistemas sequenciais, elas não são totalmente válidas, ou seja, qualquer sistema que responde a um evento dependendo das condições atuais ou do seu histórico anterior. Para este caso, utilizam-se os sistemas descritos por meio de um diagrama de estados/transições, onde um estado pode ser definido como um processo ou ação em execução e uma transição é o evento que dispara a mudança de estado nesse sistema. Normalmente, a forma de representação gráfica desse sistema é dada como uma máquina de estados finita (SILVA, 2020).

Conforme apresentado por ISTQB (2018b), componentes ou sistemas se software podem responder de maneiras diferentes a um evento, dependendo das condições

do momento ou do histórico anterior do teste. O histórico anterior pode ser definido pelo conceito de estados, como uma máquina de estados. Um diagrama de transição de estados mostra os possíveis estados do software, bem como como o software entra, sai e faz a transição entre os estados. Uma transição é iniciada por um evento, por exemplo, uma entrada de um dado pelo usuário de um valor em um campo. O evento resulta em uma transição. O mesmo evento pode resultar em duas ou mais transições diferentes do mesmo estado. A mudança de estado pode resultar no software realizando uma ação, por exemplo, emitindo um cálculo ou mensagem de erro. Os testes nessa técnica podem ser projetados para cobrir uma sequência típica de estados, para exercitar todos os estados, para exercitar cada transição, para exercitar sequências específicas de transições ou para testar transições inválidas.

No exemplo na figura 19, tem-se quatro estados e muitas possíveis transições, com o objetivo de testar o funcionamento de um LED e essa técnica poderia ser aplicada para sua verificação, porém, ficaria inviável fazê-lo de maneira manual, ainda considerando que seja um sistema simples, dessa forma, o ideal seria aplicar a técnica de maneira automatizada.

Figura 19 – Exemplo de um caso de teste de transição de estado.



Fonte: Adaptada de Silva (2020).

Seguindo as diretrizes da técnica de transição de estados, o teste pode passar por todas as transições e todos os estados, para tanto, acaba se tornando um trabalho extremamente complexo se for realizado manualmente. Assim, como em uma situação real, a complexidade seria muito maior, existiriam muitos outros estados e transições, esse tipo de teste é em sua maioria automatizado.

De acordo com BSTQB (2018), a técnica de transição de estados é usada principalmente na validação de aplicações baseadas em menus de navegação e é amplamente usado na indústria de software embarcado. Essa técnica geralmente faz uso de alguns algoritmos que podem fazer uma varredura completa da máquina de estados,

identificando todas as transições, buscando o caminho mais eficaz e curto para a realização correta dos testes. Os mais utilizados destes algoritmos são “Carteiro Chinês” (THIMBLEBY, 2003) e *Breadth-First search* (Busca amplificada) (KORF; SCHULTZE, 2005), que buscam otimizar a maneira como o teste acontece.

4.1.2 Técnicas de teste baseadas na experiência

A aplicação das técnicas de teste baseadas na experiência se dá principalmente como complemento às demais técnicas de testes funcionais (SILVA, 2020). Nela, os casos de teste são derivados da habilidade e intuição do testador e de sua experiência com aplicativos e tecnologias semelhantes. Essas técnicas podem ser úteis na identificação de testes que não foram facilmente abordados por outras técnicas mais sistemáticas e dependendo da abordagem e experiência do testador, essas técnicas podem atingir graus amplamente variados de cobertura e eficácia (ISTQB, 2018b).

4.1.2.1 Suposição de erro

Segundo Bertolino (2001), a suposição de erros é uma técnica utilizada para antecipar a ocorrência de erros, defeitos e falhas, com base no puramente no conhecimento do testador, incluindo: como a aplicação funcionou no passado, que tipo de erros tendem a ser cometidos e falhas que ocorreram em outras aplicações.

4.1.2.2 Teste exploratório

O teste exploratório consiste em aprendizagem, design e execução de teste simultâneos (BOURQUE et al., 2004). Isso significa que o teste exploratório é realizado sem casos de teste pré-especificados detalhados, ou seja, testes sem script. A abordagem teste exploratório pode ser usada para diferentes tipos de teste usando várias técnicas ou métodos (ITKONEN; RAUTIAINEN, 2005). É uma técnica informal, ou seja, não há um procedimento definido para sua realização, tendo como objetivo fazer com que se conheça mais a respeito do componente a ser testado. Em algumas ocasiões, o teste exploratório é realizado usando testes baseados em sessões para estruturar as atividades de teste. Essa técnica é muito útil quando há poucas ou inadequadas

especificações de requisitos ou prazos realmente muito curtos (BSTQB, 2018).

4.1.2.3 Teste baseado em checklist

De acordo com Silva (2020), essa é uma técnica similar ao Teste exploratório, porém, traz uma formalidade ao incluir uma lista de alto nível, contendo as mínimas funções ou funcionalidades a serem testadas, mesmo que não exista ainda um procedimento de teste especificado. Em testes baseados em checklist, os profissionais responsáveis pelos testes projetam, implementam e executam testes para cobrir as condições de teste encontradas em uma lista de verificação. Essas checklists podem ser construídas com base na experiência, no conhecimento sobre o que é importante para o usuário ou na compreensão do porquê e como o software falha (BSTQB, 2018). Na ausência de casos de teste detalhados, essa técnica pode fornecer diretrizes e um grau de consistência. Como essas são listas de alto nível, é provável que ocorra alguma variabilidade no teste real, resultando em uma cobertura potencialmente maior, mas menos repetibilidade (BSTQB, 2018).

4.2 TÉCNICAS DE TESTES CAIXA-BRANCA

Segundo Neto e Dias (2007), o teste de caixa branca abrange testes com base na estrutura interna do sistema analisado. E essa estrutura pode englobar vários itens, como código, design e fluxos de informações dentro do sistema de software testado. A sua eficácia pode ser medida por meio da cobertura estrutural do software, considerando quanto da estrutura do componente de software (funções, decisões e instruções) foi abordado no teste, esse valor é expresso em porcentagem. Para a elaboração de seguinte execução das técnicas de teste caixa-branca, pode ser necessário que a pessoa responsável possua certas habilidades ou conhecimentos específicos, por exemplo, ter noção sobre a forma como o código foi construído, saber utilizar ferramentas de cobertura de testes e interpretar de maneira correta os resultados obtidos.

As técnicas de teste de caixa-branca podem ser aplicadas em todos os níveis de teste (caixa-cinza), mas as técnicas relacionadas ao código apresentadas adiante são mais comumente aplicadas ao nível de teste de componente e integração. Essas

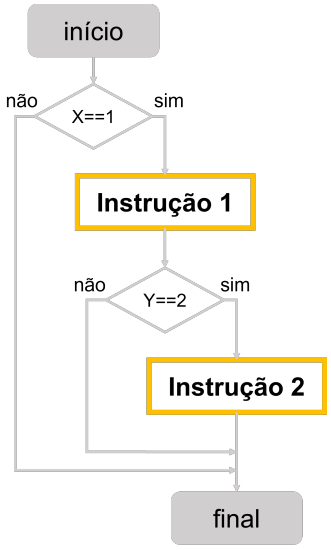
técnicas possuem algumas características em comum, de acordo com BSTQB (2018), são elas:

- Condições, casos e dados de teste são derivados de uma base de teste que pode incluir código, design de software, design detalhado ou qualquer outra fonte de informação sobre a estrutura do software.
- A cobertura é medida com base nos itens testados dentro de uma estrutura selecionada (por exemplo, o código ou interfaces específicas) e a técnica aplicada ao teste.

4.2.1 Teste de cobertura de instrução

O teste de Cobertura de instrução exercita as declarações (instruções) executáveis em potencial no código do componente. A cobertura é medida como o número de instruções executadas pelos testes dividido pelo número total de instruções executáveis no objeto de teste, normalmente expresso como uma porcentagem (BEYDEDA; GRUHN, 2001). Nessa porcentagem, qualquer valor inferior a 100% de cobertura de instrução significa que nem todas as linhas de código foram executadas. A vantagem de medir a cobertura de instrução é que ela é capaz de isolar a parte do código que não pôde ser executada.

Figura 20 – Exemplo de um teste de cobertura de instrução.



Fonte: Adaptada de Silva (2020).

Na figura 20, pode-se observar de maneira simplificada como essa técnica acontece em um trecho de código específico. Verifica-se se as instruções 1 e 2 estão

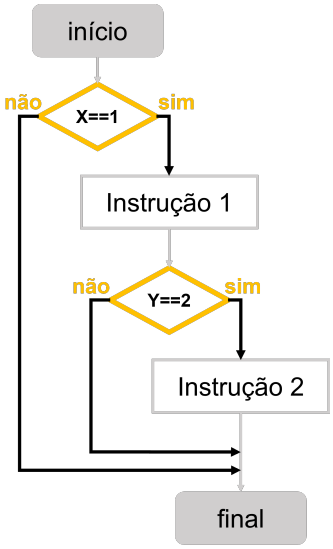
ocorrendo, dependendo dos valores de X e Y. Para esse exemplo simplificado, a cobertura seria de 100%, caso X==1 e Y==2, ou seja, com apenas um caso de teste. Hoje em dia, existem softwares que realizam esse tipo de teste e ao final podem geral relatórios com quais instruções foram executas e quais não foram, relatando detalhes para cada caso.

4.2.2 Teste de cobertura de decisão

Essa técnica tem o objetivo de testar todas as opções (verdadeiras ou falsas) em cada instrução de controle que também inclui a decisão composta (quando a segunda decisão depende da decisão anterior) (ROBSON, 2009). A cobertura é medida como o número de resultados de decisão executados pelos testes dividido pelo número total de resultados de decisão no objeto de teste, normalmente expresso como uma porcentagem (BSTQB, 2018).

Na figura 21, seguimos o mesmo exemplo utilizado na técnica anterior, onde agora são verificadas as condições das decisões (IF). Com esse caso, a decisão que verifica o valor de Y só ocorrerá no caso da decisão que verifica o valor de X ser verdadeira. Portanto, seriam necessários três casos de teste para que uma cobertura de 100% aconteça.

Figura 21 – Exemplo de um teste de cobertura de decisão.

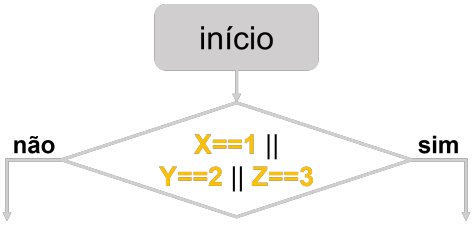


Fonte: Adaptada de Silva (2020).

4.2.3 Condição modificada e cobertura de decisão (MC/DC)

Essa técnica é uma extensão da cobertura de decisão, nela ocorre a verificação das condições dentro das decisões. Quando 100% de cobertura de decisão é alcançada, o teste executa todos os resultados de decisão, o que inclui testar o resultado verdadeiro e o resultado falso, mesmo quando não há declaração falsa explícita dentro da decisão testada. A cobertura de decisão ajuda a encontrar defeitos no código onde outros testes não obtiveram resultados verdadeiros e falsos, já que pode haver algum erro na especificação das condições (ROBSON, 2009).

Figura 22 – Exemplo de um teste de condição modificada e cobertura de decisão.



Fonte: Adaptada de Silva (2020).

Quadro 8 – Exemplo de um teste de condição modificada e cobertura de decisão.

X==1	Y==2	Z==3	Resultado
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Fonte: Autoria própria (2021).

No exemplo da figura 22, tem-se um IF, que resultará 1 (sim) no caso de qualquer das condições ocorrer (X==1 || Y==2 || Z==3). No quadro 8, tem-se todas as possibilidades decisão. Como há três condições, serão necessários oito casos de teste para cobertura de 100% da condição.

5 RESULTADOS E DICUSSÕES

As técnicas de caixa-branca geralmente são realizadas pela empresa fornecedora de software, portanto o seu processo é avaliado e qualificado segundo o ASPICE e deve seguir os requisitos de teste da ISO 26262.

As técnicas abordadas nesse trabalho podem ser aplicadas em todos os subsistemas eletrônicos do veículo, que englobam muitas ECUs (Powertrain, chassi, carroceria e multimídia). As técnicas são aplicadas por ECU, portanto, deve haver uma equipe destinada a cada subsistema, sendo capaz de seguir da melhor maneira possível as recomendações processos de ASIL.

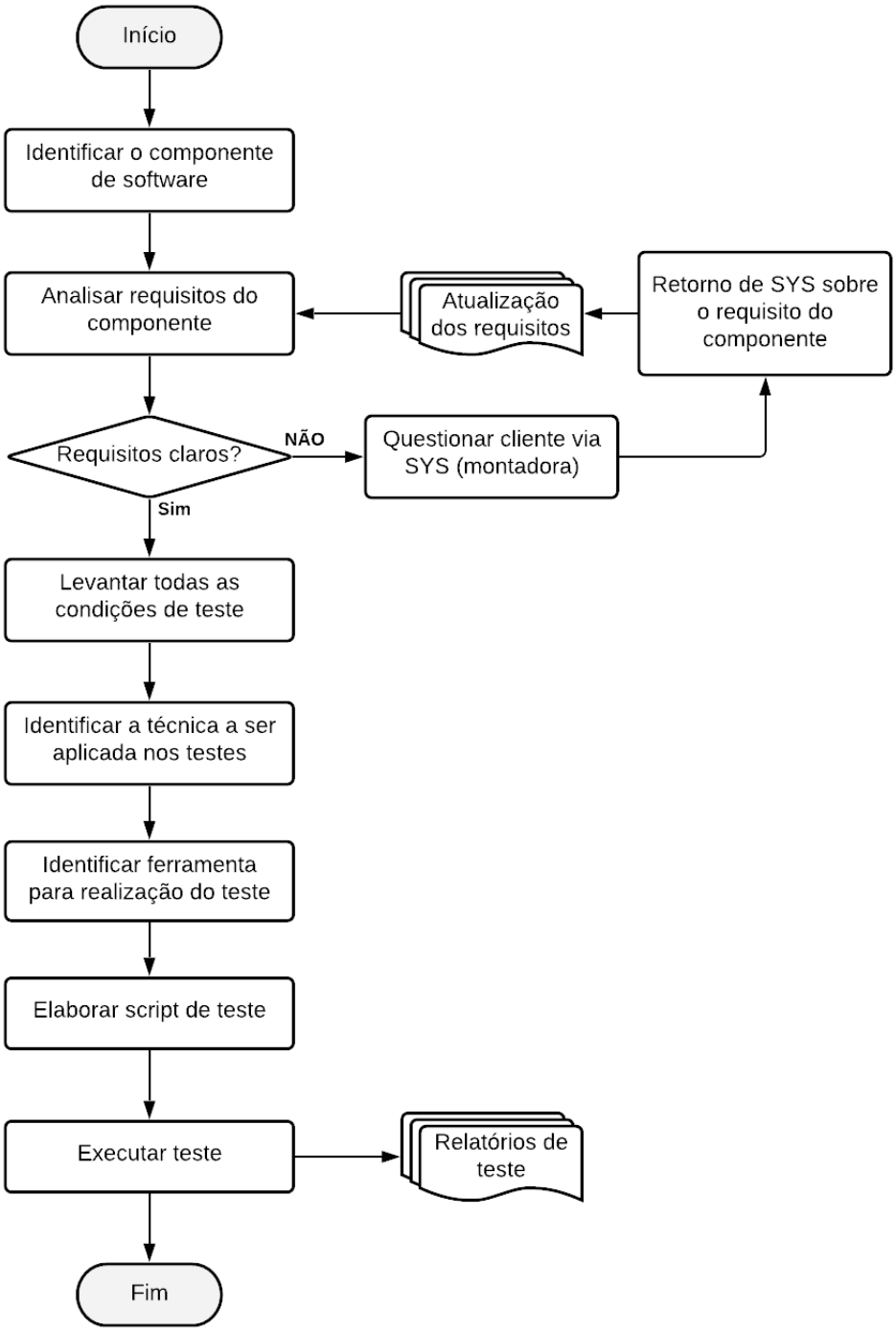
Depois do estudo das técnicas de testes caixa-branca e do ciclo de desenvolvimento de produto automotivo, elaborou-se uma proposta de fluxo de ações que seriam necessárias para execução testes em ECUs utilizando essa categoria de técnicas. Estão representadas no fluxograma 1, para isso, considera-se que são realizados pelo fornecedor de software, ainda em versão interna antes de disponibilização oficial ao cliente (montadora). Esse fluxo, poderia ser aplicado nos níveis de teste de componentes ou nos testes de integração.

A primeira ação refere-se à identificação do(s) componente(s) de software, nessa etapa é feita a delimitação de quais funcionalidades do veículo serão consideradas. Por exemplo, em uma ECU para o painel de instrumentos do veículo (IPC - *Instrument Panel Cluster*), pode haver componentes como *Seat Belt Reminder*, *Cruise Control*, *Human-Machine Interface* entre outros. Para que haja um controle do que será testado, essa etapa é muito importante.

Em seguida, tem-se a análise dos requisitos para o(s) componente(s) de software identificado(s) na etapa anterior. Nessa fase, pode ser que surjam algumas dúvidas técnicas específicas a respeito de como o componente deve se comportar, portanto, é de extrema importância que haja um canal facilitado para esclarecimentos entre o time de SYS do fornecedor e o cliente, assim, evita-se que venham a ocorrer problemas futuros. Então se um requisito está incompleto, ambíguo ou que cause alguma dúvida, o cliente é notificado e envia uma nova versão dos requisitos, para que assim possa se dar sequência aos testes.

Com os requisitos clarificados em mãos, o testador então identifica os cenários possíveis de teste se é um sistema combinatório ou sequencial. Existem no mercado

Fluxograma 1 – Fluxo e ações proposto para testes caixa-branca.



Fonte: Autoria própria (2021).

alguns softwares que podem facilitar essa tarefa, pois quando se trata de um componente muito complexo, essa atividade pode se tornar realmente difícil. Em seguida, o testador já é capaz de identificar qual técnica necessita ser aplicada (podendo ser mais de uma). Com isso, já pode ter uma noção de qual ferramenta vai lhe favorecer mais na hora de executar o teste.

Com a técnica e a ferramenta de testes definidas, o testador pode então seguir para a elaboração do script, levando em consideração todos os cenários de testes necessários. Em seguida, ocorre a execução dos testes propriamente dita, essa etapa pode ser mais demorada de acordo com a complexibilidade das funções que serão analisadas durante o teste.

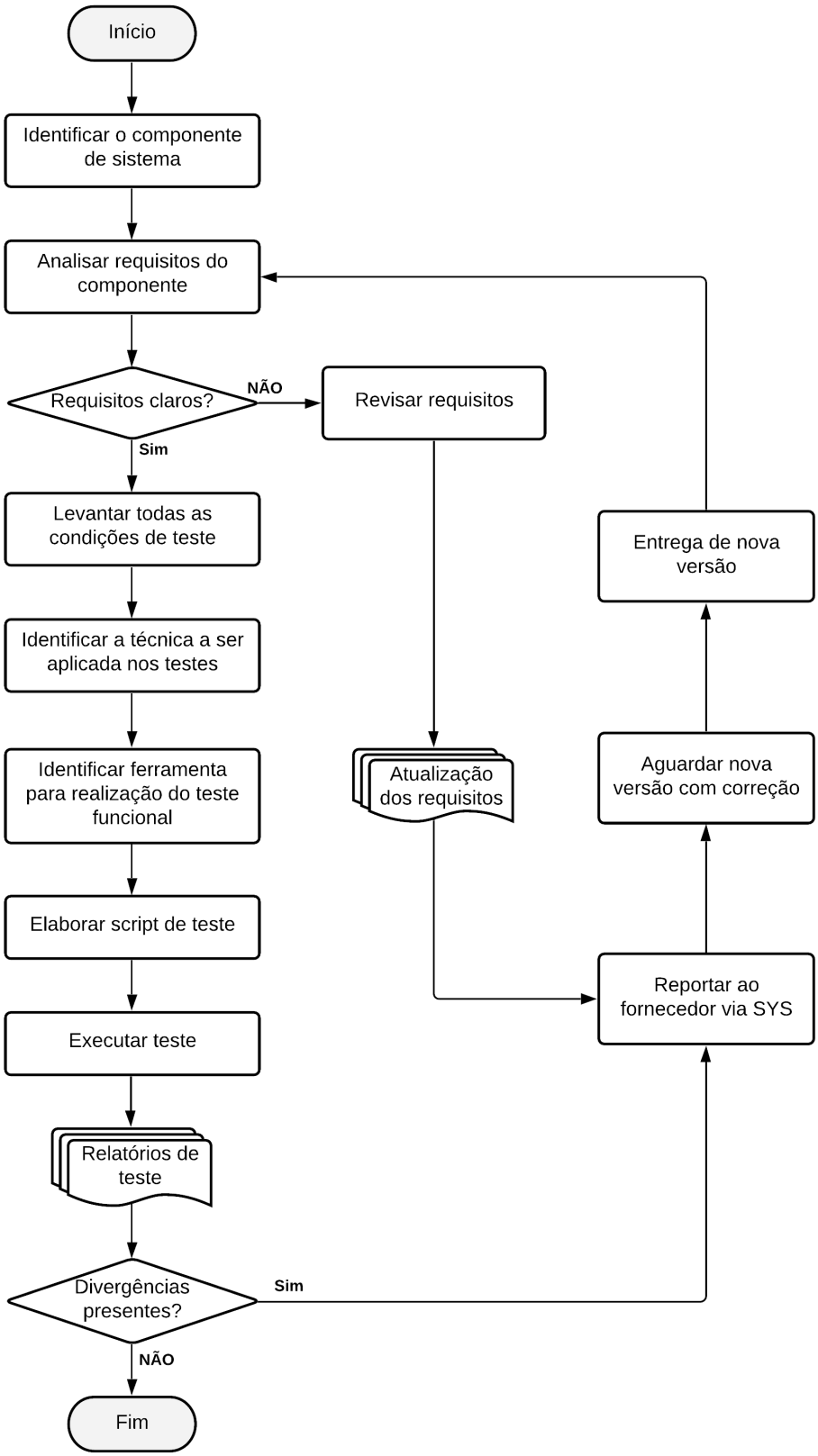
Ao final da execução, são gerados relatórios dos testes, podendo mudar de acordo com as ferramentas e técnicas utilizadas. Esse relatório pode conter informações como: cobertura do teste, se está de acordo com os requisitos ou não entre outras informações que podem ser de interesse gerencial para negociações. A partir desse ponto, o testador encerra suas atividades. Caso ocorra algum problema reportados nos relatórios de testes, esse problema é corrigido em uma outra versão do software e então todo o processo se repete.

Para os testes funcionais, o processo é muito parecido, porém, agora considerou-se que esses testes seriam realizados já pela montadora, com a ECU realmente funcional. O fluxo de trabalho proposto está ilustrado no fluxograma 2.

A primeira etapa é muito similar à primeira do fluxograma 1, porém, os componentes a serem identificados são os de sistema, a fim de se verificar quais funcionalidades devem ser esperadas da ECU.

Então, segue-se para a análise dos requisitos para buscar entender “o quê” a ECU deve fazer, com base nos inputs que receber. Se os requisitos não estiverem claros, e isso só for notado já quando o software embarcado foi entregue ao cliente, então deve ser feita uma revisão dos requisitos. Essa revisão será repassada ao fornecedor, para que então as correções possam ser implementadas. Até que haja um retorno do fornecedor com uma nova versão para testes para então dar sequência ao processo. As etapas de levantar as condições de teste, identificar a técnica mais adequada a ser aplicada, identificar a ferramenta ideal para sua realização, elaboração do script de teste e sua execução, são muito parecidas ao processo para testes caixa-branca, porém, agora considerando as técnicas de testes funcionais.

Fluxograma 2 – Fluxo e ações proposto para testes funcionais.



Fonte: Autoria própria (2021).

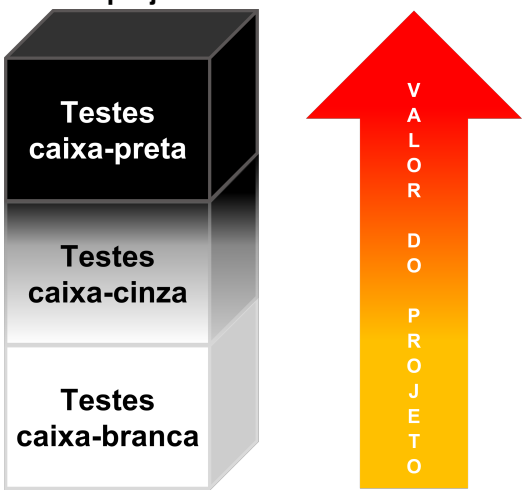
Ao final da execução do teste, são gerados relatórios, onde encontram-se detalhes a respeito das condições de teste e os resultados. Se os resultados estiverem de acordo com os requisitos funcionais, então finaliza-se o ciclo de teste para o componente em questão, porém, se forem observadas divergências, então são reportadas ao fornecedor por meio de SYS novamente. Depois disso, aguarda-se uma nova entrega para realização do teste novamente para o componente de sistema em questão.

Para ambas as categorias de testes, existem softwares no mercado que podem facilitar muito o processo de trabalho de um engenheiro ou engenheira de sistema que é responsável pela elaboração do teste. Isso pode ser uma questão que valha a pena levar a nível de gestão, já que as horas de engenharia podem ser otimizadas.

Com essa análise realizada a respeito das técnicas de testes funcionais e de caixa-branca executados em unidades eletrônicas de controle aplicadas em automóveis, observa-se que quanto mais alto o nível de testes no ciclo V, mais ele acaba se tornando obrigatório, segundo a ISO 26262. Observa-se também que quando mais cedo um defeito for encontrado, menor será o impacto ao longo do projeto, levando em consideração o aspecto financeiro (horas de engenharia e equipamentos). Com essa conclusão, pode-se então dizer que o nível de testes unitários podem ser os de maior impacto e são os que devem ter maior atenção, já que se a maioria dos defeitos forem detectados nele, o projeto terá menos erros reportados adiante, portanto, menos retrabalho.

Essa relação pode ser representada na figura 23, onde o valor do projeto tende a crescer quando os defeitos são encontrados em níveis mais altos do projeto.

Figura 23 – Crecimento do valor do projeto de acordo com os testes.



Fonte: Autoria própria (2021).

Em suma pode-se destacar algumas das mais relevantes diferenças entre os tipos de técnicas citadas, elas estão presentes no quadro 9.

Quadro 9 – Comparação entre testes Caixa-Branca e Caixa-Preta.

TESTES CAIXA-BRANCA	**TESTES CAIXA-PRETA**
Análise Estrutural	Análise Funcional
Custo extra menor casos os defeitos sejam detectados	Alto impacto em custo extra caso defeitos sejam detectados
Requer conhecimento específico de software.	Não requer conhecimento específico de software.
Testa software	Testa hardware completo

Fonte: Autoria própria (2021).

Essas informações devem ser levadas em consideração em toda tomada de decisão necessária durante os processos de desenvolvimento e validação de softwares automotivos, já que uma ECU testada de maneira inapropriada, sendo aplicada a técnica não ideal ou de maneira incorreta, pode acarretar em grandes perdas a todos os envolvidos.

6 CONCLUSÕES

Testes são artifícios obrigatórios para a elaboração adequada de qualquer projeto, seja de software ou não, pois garantem que os requisitos foram cumpridos, que as correções necessárias foram realizadas corretamente e pode prevenir futuras falhas que venham a ser identificadas em um nível superior do desenvolvimento do produto, ou até mesmo, no pior dos casos, já no usuário final.

Na indústria automotiva não é diferente, já que o produto final, que são os veículos, podem colocar em risco vidas humanas caso não estejam funcionando de maneira adequada. Portanto, é de extrema importância que os testes desenvolvidos para aplicação nessa área sigam as metodologias corretas, obedecendo às regulamentações em vigor e aos requisitos definidos, depois de estudos, pelas montadoras, abrangendo o máximo de condições possíveis que garantam a segurança e o bom funcionamento no veículo.

Fica evidente então a necessidade cada vez maior da capacitação dos profissionais de engenharia para se adequarem às necessidades demandadas pelas empresas que desenvolvem softwares para ECUs, já que este é um campo ainda com muitas lacunas a serem preenchidas no decorrer da formação de engenheiros que direcionam suas carreiras a esse mercado, que possui uma demanda de profissionais em ascensão.

A capacidade de tomada de decisão a respeito de quais técnicas de teste devem ser aplicadas de acordo com os níveis de desenvolvimento do produto, são fortemente embasadas em normas como a ISO 26262. Dessa forma, cabe ao profissional ter conhecimento para poder fazer a interpretação correta a respeito da situação que lhe é apresentada. Assim, cada vez mais, os processos para desenvolvimento se tornam mais confiáveis, trazendo benefícios ao profissional responsável e à empresa fornecedora do produto, já que o nível de confiabilidade de seus processos cresce, aumentando seu conceito, tornando-a mais procurada pelas montadoras, trazendo uma vantagem competitiva.

Os objetivos previamente estabelecidos para a realização desse trabalho foram alcançados com sucesso, trazendo muito conhecimento de uma área ainda com alto potencial de estudo para engenharia. Fica como sugestão para trabalhos futuros os seguintes temas:

- Análise das ferramentas para testes caixa-branca e/ou funcionais em ECUs;

- Análise de linguagem para scripts de testes estruturais em software automotivo;
- Abordagens para testes funcionais de ECUs em veículos autônomos.

REFERÊNCIAS

ABBOTT-MCCUNE, S.; SHAY, L. A. Intrusion prevention system of automotive network CAN bus. In: IEEE. 2016 IEEE International Carnahan Conference on Security Technology (ICCST). [S.l.: s.n.], 2016. P. 1–8.

ABOAGYE, A. et al. Facing digital disruption in mobility as a traditional auto player. **Economia e Sociedade, Campinas, Unicamp. IE**, v. 20, 2017.

AHSAN, M.; STOYANOV, S.; BAILEY, C. Prognostics of automotive electronics with data driven approach: A review. **2016 39th International Spring Seminar on Electronics Technology (ISSE)**, p. 279–284, 2016.

ALAM, M. S. U. **Securing vehicle Electronic Control Unit (ECU) communications and stored data**. 2018. Tese (Doutorado) – Queen's University (Canada).

ASPICE. **Automotive SPICE Introduction**. [S.l.: s.n.], 2018. Quality Management Center in the German Association of Automotive Industry (VDA QMC). Disponível em: http://www.automotivespice.com/about/. Acesso em: 8 mar. 2021.

ASPICE. **Automotive SPICE Process Assessment / Reference Model - German Association of the Automotive Industry (VDA)**. [S.l.: s.n.], 2017. QMC Working Group 13 / Automotive SIG, Version 3.1. Disponível em: http://www.automotivespice.com/download/. Acesso em: 8 mar. 2021.

BEIZER, B. **Software testing techniques**. [S.l.]: Dreamtech Press, 2003.

BEREISA, J. Applications of microcomputers in automotive electronics. **IEEE Transactions on Industrial Electronics**, IEEE, n. 2, p. 87–96, 1983.

BERTOLINO, A. Software testing. **SWEBOK**, p. 69, 2001.

BEYDEDA, S.; GRUHN, V. Integrating white-and black-box techniques for class-level regression testing. In: IEEE. 25TH Annual International Computer Software and Applications Conference. COMPSAC 2001. [S.l.: s.n.], 2001. P. 357–362.

BHUVANESWARI, T.; PRABAHARAN, S. A survey on software development life cycle models. **International Journal of Computer Science and Mobile Computing**, v. 2, n. 5, p. 262–267, 2013.

BLACK, R. **Managing the testing process**. [S.l.]: John Wiley & Sons, 2009.

BOSCH, R. G. U. A. A. **Bosch Automotive Electrics and Automotive Electronics: Systems and Components, Networking and Hybrid Drive**. [S.l.]: Springer Vieweg, 2014.

BOURQUE, P. et al. Guide to the software engineering body of knowledge. **IEEE**, 2004.

BSTQB, B. S. T. Q. B. **Certified Tester Foundation Level Syllabus**. [S.l.: s.n.], 2018. Version V3.1. Disponível em: https://bstqb.org.br/b9/doc/syllabus_ctfl_2018br.pdf. Acesso em: 28 fev. 2021.

BUCANAC, C. **The V-Model**. [S.l.: s.n.], jan. 1999. Quality Management, DPT404. Disponível em: http://www.bucanac.com/documents/The_V-Model.pdf. Acesso em: 25 fev. 2021.

CAMINSKI, L. ISO 26262: segurança funcional no desenvolvimento de sistemas automotivos. Universidade Tecnológica Federal do Paraná, 2018.

CHRISTMANN, E. Data Communication in the Automobile – Part 1: Architecture, Tasks, and Advantages of Serial Bus Systems. Vector Informatik GmbH, 2017.

CHRISTMANN, E. Data Communication in the Automobile – Part 2: Reliable Data Exchange with CAN. Vector Informatik GmbH, 2017.

CORTESINI, L. **Afinal o que é ECU? Como elas funcionam?** [S.l.: s.n.], 17 abr. 2017. Flat-Out. Disponível em: https://flatout.com.br/afinal-o-que-e-ecu-como-elas-funcionam/. Acesso em: 18 abr. 2021.

EBERT, C.; FAVARO, J. Automotive software. **IEEE Annals of the History of Computing**, IEEE Computer Society, v. 34, n. 03, p. 33–39, 2017.

FORSBERG, K.; MOOZ, H. The relationship of system engineering to the project cycle. In: WILEY ONLINE LIBRARY, 1. INCOSE International Symposium. [S.l.: s.n.], 1991. v. 1, p. 57–65.

FOWLER, M. Put your process on a diet software development. **Retrieved Sept**, v. 22, p. 2004, 2000.

HIGHSMITH, J. A.; HIGHSMITH, J. **Agile software development ecosystems**. [S.l.]: Addison-Wesley Professional, 2002.

HOODA, I.; CHHILLAR, R. S. Software test process, testing types and techniques. **International Journal of Computer Applications**, Citeseer, v. 111, n. 13, 2015.

ISO. **ISO/FDIS 26262**: Road vehicles - Functional safety. [S.l.], 2018. 593 p.

ISTQB. **Foundation Level Specialist CTFL Automotive Software Tester (CTFL-AuT)**. [S.l.: s.n.], jul. 2018. Version (2.0.2). Disponível em: https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html. Acesso em: 28 fev. 2021.

ISTQB. **International Software Testing Qualifications Board - Certified Tester Foundation Level Syllabus CTFL**. [S.l.: s.n.], 2018. Version 3.1. Disponível em:

https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html. Acesso em: 28 fev. 2021.

ISTQB. **International Software Testing Qualifications Board - Certifying Software Testers Worldwide**. [S.l.: s.n.], 2020. ISTQB. Disponível em: https://www.istqb.org. Acesso em: 10 mar. 2021.

ITKONEN, J.; RAUTIAINEN, K. Exploratory testing: a multiple case study. In: IEEE. 2005 International Symposium on Empirical Software Engineering, 2005. [S.l.: s.n.], 2005. 10–pp.

JAIKAMAL, V. **Model-based ecu development–an integrated mil-sil-hil approach**. [S.l.], 2009.

JORGENSEN, P. C. **Software testing: a craftsman's approach**. [S.l.]: CRC press, 2018.

JOVANOVIĆ, I. Software testing methods and techniques. **The IPSI BgD Transactions on Internet Research**, v. 30, 2006.

KANER, C.; PADMANABHAN, S.; HOFFMAN, D. **The Domain Testing Workbook**. [S.l.]: Context Driven Press, 2013.

KLOPPER, R.; GRUNER, S.; KOURIE, D. G. Assessment of a framework to compare software development methodologies. In: PROCEEDINGS of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries. [S.l.: s.n.], 2007. P. 56–65.

KORF, R. E.; SCHULTZE, P. Large-scale parallel breadth-first search. In: AAAI. [S.l.: s.n.], 2005. v. 5, p. 1380–1385.

LARMAN, C.; BASILI, V. R. Iterative and incremental developments. a brief history. **Computer**, IEEE, v. 36, n. 6, p. 47–56, 2003.

LIU, B.; ZHANG, H.; ZHU, S. An incremental v-model process for automotive development. In: IEEE. 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). [S.l.: s.n.], 2016. P. 225–232.

LUO, L. Software testing techniques. **Institute for software research international Carnegie mellon university Pittsburgh, PA**, v. 15232, n. 1-19, p. 19, 2001.

MISHRA, A.; DUBEY, D. A comparative study of different software development life cycle models in different scenarios. **International Journal of Advance research in computer science and management studies**, v. 1, n. 5, 2013.

MONITORA, E. **Quais os tipos de testes de software e por que automatizá-los?** [S.l.: s.n.], 11 fev. 2019. Monitora Soluções Tecnológicas. Disponível em: https://www.monitoratec.com.br/blog/quais-os-tipos-de-testes-de-software-e-por-que-automatiza-los/. Acesso em: 12 abr. 2021.

MYERS, G. J.; BADGETT, T. et al. **The art of software testing**. [S.l.]: Wiley Online Library, 2004. v. 2.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 2011.

NEEMEH, S. **What is ASPICE in Automotive?** [S.l.: s.n.], jun. 2020. LHP Engineering Solutions. Disponível em: https://www.lhpes.com/blog/what-is-aspice-in-automotive. Acesso em: 5 mar. 2021.

NETO, A.; DIAS, C. Introdução a teste de software. **Engenharia de Software Magazine**, v. 1, p. 22, 2007.

NIBERT, J.; HERNITER, M. E.; CHAMBERS, Z. Model-based system design for MIL, SIL, and HIL. **World Electric Vehicle Journal**, Multidisciplinary Digital Publishing Institute, v. 5, n. 4, p. 1121–1130, 2012.

ONUMA, Y.; TERASHIMA, Y.; KIYOHARA, R. ECU software updating in future vehicle networks. In: IEEE. 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA). [S.l.: s.n.], 2017. P. 35–40.

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. [S.l.]: Palgrave macmillan, 2010.

RIBEIRO, R. **Quantos módulos eletrônicos existem em um automóvel de luxo?** [S.l.: s.n.], 20 abr. 2018. Quatro Rodas. Disponível em: https://quatrorodas.abril.com.br/auto-servico/quantos-modulos-eletronicos-existem-em-um-automovel-de-luxo/. Acesso em: 15 abr. 2021.

ROBSON, S. White Box Testing. **Software Education**, v. 1, p. 15, 2009.

ROSA, J. G. **Grande sertão: veredas**. [S.l.]: Livraria José Olimpio Editora, Rio de Janeiro, 1956.

ROYCE, W. W. Managing the development of large software systems. proceedings of IEEE WESCON. **Los Angeles**, p. 328–388, 1970.

RUPARELIA, N. B. Software development lifecycle models. **ACM SIGSOFT Software Engineering Notes**, ACM New York, NY, USA, v. 35, n. 3, p. 8–13, 2010.

SCHAUFFELE, J.; ZURAWKA, T. **Automotive software engineering**. [S.l.]: Springer, 2010.

SCHWABER, K.; BEEDLE, M. **Agile software development with Scrum**. [S.l.]: Prentice Hall Upper Saddle River, 2002. v. 1.

SILVA, J. X. **Aplicando técnicas de Teste Funcional com o vTESTstudio**. [S.l.: s.n.], 24 jun. 2020. Vector Informatik GmbH. Disponível em: https://www.vector.com/br/pt/. Acesso em: 2 mar. 2021.

THIMBLEBY, H. The directed chinese postman problem. **Software: Practice and Experience**, Wiley Online Library, v. 33, n. 11, p. 1081–1096, 2003.

VULI, P.; BADALAMENT, M.; JAIKAMAL, V. **Maximizing test asset re-use across MIL, SIL, and HIL development platforms**. [S.l.], 2010.

ÍNDICE REMISSIVO

ASIL, 39–42, 59
ASPICE, viii, x, 14, 33, 35–39, 41
AUTOSAR, 11

BP, 38

CAN, viii, 11, 19–21
CAPL, 13
CL, 36
CT, 47, 48

ECU, viii, 12, 14–21, 23, 30, 31, 34, 35, 39–41, 43, 46, 47, 50, 59, 61, 64–66

GP, 38

HIL, 34, 35

IPC, 59
ISO, x, 14, 15, 33, 39–42, 59, 63
ISTQB, x, 41

LED, 53
LIN, 11, 20

MBT, x, 33
MC/DC, x, 58
MIL, 34
MOST, 11, 20

PA, 38
PAM, 36
PC, 34
PRM, 36

SAE, 20
SIL, 34

UTFPR, i

WP, 38