UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

Relatório de estágio: Desenvolvimento de metodologia de *Quality Assurance* para projetos de sistemas embarcados

Yago Luiz Monteiro Silva

Campina Grande, PB

Dezembro de 2018

Yago Luiz Monteiro Silva

Relatório de estágio: Desenvolvimento de metodologia de *Quality Assurance* para projetos de sistemas embarcados

Relatório de estágio apresentado à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Área de Concentração: Sistemas Embarcados

Orientador: Gutemberg Gonçalves dos Santos Júnior

Campina Grande, PB

Dezembro de 2018

Yago Luiz Monteiro Silva

Relatório de estágio: Desenvolvimento de metodologia de *Quality Assurance* para projetos de sistemas embarcados

Aprovado em ___/___ /___

Professor Avaliador

Universidade Federal de Campina Grande

Avaliador

Gutemberg Gonçalves dos Santos Junior

Universidade Federal de Campina Grande

Orientador

Agradecimentos

Agradeço inicialmente a Igor Pinheiro e ao Professor Gutemberg pela oportunidade de realização do estágio, por acreditar em mim e no meu trabalho.

A Bruno Winkeler e Lorena Maia pelo apoio técnico e por possibilitarem o desenvolvimento das etapas do trabalho.

Finalmente, aos colegas da célula Tokyo pelas discussões e por construírem, em parceria, a base de conhecimento em qualidade.

Lista de Figuras

Lista de Tabelas

Conteúdo

1 Introdução

Este trabalho tem como intuito descrever as atividades realizadas no âmbito profissional durante o Estágio Supervisionado, como parte indispensável para a formação acadêmica em Engenharia Elétrica. O estágio foi realizado durante o período de 22/10/2018 a 05/12/2018 no Embedded/Virtus/UFCG com carga horária semanal de trinta (30) horas semanais e atendendo aos requisitos previstos na Resolução 01/2012 do Colegiado do Curso de Graduação de Engenharia Elétrica e em consonância com a Lei do Estágio (11.788/2008).

1.1 O Laboratório

O Laboratório de Sistemas Embarcados e Computação Pervasiva, alocado no Centro de Engenharia Elétrica e Informática da UFCG (CEEI) foi fundado em Dezembro de 2005 e em 2015, deu origem ao Virtus, que é um órgão suplementar ao Embedded com foco em desenvolvimento e inovação.

A missão do Embedded é avançar no estado da arte nas áreas de sistemas embarcados e computação pervasiva promovendo, a partir dos convênios da Lei da Informática, soluções de alta tecnologia para satisfazer as necessidades práticas da indústria, avançando tanto no campo da ciência quanto no *know how* da implementação dessas soluções.

2 Área de atuação

As atividades desenvolvidas durante o período do estágio compreendem o desenvolvimento de um processo de validação funcional de projetos de natureza de *software* embarcado dentro do contexto dos projetos desenvolvidos pelo Embedded/Virtus.

O desenvolvimento de sistemas, sejam de natureza de *Hardware* ou *Software*, estão sujeitos a erros de desenvolvimento e/ou erros de projeto devido a diversos fatores. Em qualquer uma das situações, um erro não descoberto durante a fase de projeto pode gerar falhas catastróficas e prejuízos enormes à empresa que lançou esse produto. Em outras palavras, quanto antes no ciclo de vida do projeto um erro for descoberto, menor é o custo do seu reparo.

Figura 2.1: Custo da descoberta do erro e a fase do projeto

Nesse contexto, se insere o conjunto de medidas chamadas de ”*Quality Assurance*”, que consistem em garantir, por meio do processo contínuo de validação e verificação, garantias aos clientes de que o produto entregue possui o mínimo de erros possível dentro do conjunto de especificações do projeto.

3 Avaliação de Qualidade

De modo geral, a conotação da palavra qualidade está sempre atrelada à satisfação que o objeto/serviço classificado proporciona a quem o solicitou, ou seja, qualidade denota o grau em que o produto cumpre o propósito para o qual foi designado.

Em uma visão mais técnica, a ISO (International Organization for Standartization), por meio da ISO 9000 (segunda edição, 2000), define qualidade como ”o grau em que um conjunto de características de um produto ou serviço satisfaz um conjunto de requisitos previamente estabelecidos”.

3.1 Avaliação de Qualidade em *Software*

Para produtos de natureza de *Software*, a métrica de avaliação de qualidade é padronizada e definida pela ISO 25010 de 2011. Essa norma define um conjunto de características e métricas necessárias para aprovação de um produto de *software* sob a etiqueta de um modelo de qualidade.

O modelo de qualidade de *software* está disposto em oito categorias: Adequação Funcional, Performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança da informação, Manutenabilidade e Portabilidade que serão detalhadas a seguir.

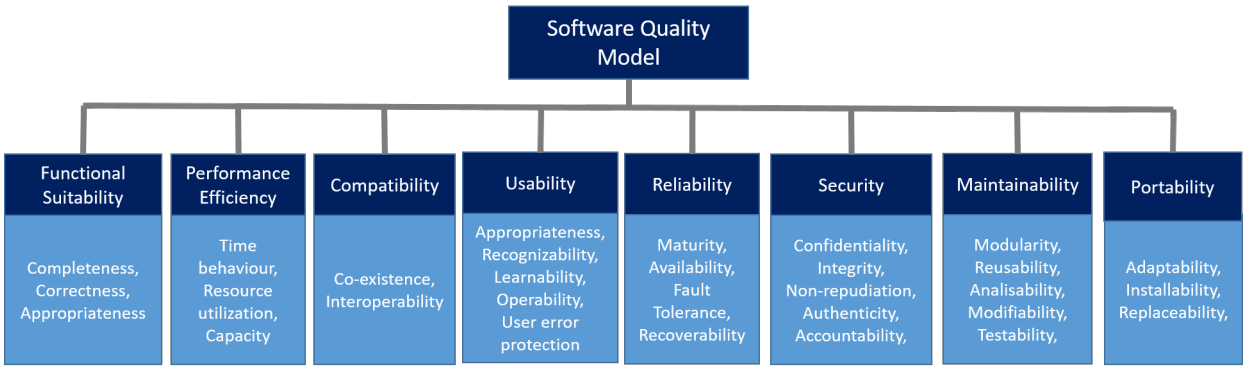

Figura 3.1: Modelo de qualidade de Software definido pela ISO 25010

3.1.1 Adequação Funcional

Essa característica representa o grau em que um produto ou sistema provê funcionalidades capazes de satisfazer o propósito de uso da entidade sob determinadas condições de uso. Essa característica é adquirida quando as subcaracterísticas a seguir são respeitadas:

- Completude funcional: grau em que o conjunto de funções desenvolvidas cobrem todas as tarefas pré-estabelecidas e objetivos do usuário;
- Corretude funcional: grau em que o produto ou sistema devolve as respostas corretas para um dado nível pré-estabelecido de precisão;
- Adequação funcional: grau em que as funções facilitam o cumprimento das tarefas e objetivos do sistema;

3.1.2 Performance

Essa característica representa a performance relativa ao conjunto de recursos usados sob condições pré-estabelecidas. Essa característica é adquirida quando as subcaracterísticas a seguir são respeitadas:

- Comportamento temporal: grau em que os tempos de resposta, processamento e *throughput* de um sistema são respeitados quando esse sistema está funcionando conforme os requisitos;
- Utilização de recursos: grau em que a quantidade e os tipos de recursos usados por um produto sob operação normal estão de acordo com os requisitos;
- Capacidade: grau em que os limites extremos de operação de um sistema estão de acordo com os requisitos;

3.1.3 Compatibilidade

Essa característica representa o grau em que um produto, sistema ou componente é capaz de trocar informações com outros produtos, sistemas ou componentes enquanto usam o mesmo ambiente de hardware ou software. Essa característica é garantida quando as subcaraterísticas a seguir são respeitadas:

- Co-existência: grau em que um produto é capaz de funcionar eficientemente enquanto compartilha o ambiente e recursos com outros produtos sem impacto negativo no funcionamento do todo;
- Interoperabilidade: grau em que dois ou mais sistemas, produtos ou componentes são capazes de trocar informação e usar a informação que foi trocada;

3.1.4 Usabilidade

Grau em que um produto ou sistema pode ser usado para atingir os objetivos com eficiência, eficácia e satisfação em um determinado contexto de uso. Essa característica é atingida quando as subcaracterísticas são respeitadas:

- Reconhecibilidade: grau em que os usuários conseguem identificar se o sistema ou produto é apropriado para suas necessidades;
- Facilidade de aprendizado: grau em que o usuário é capaz de atingir determinados objetivos de aprendizado com eficiência, eficácia e satisfação;
- Operabilidade: Grau em que o sistema dispõe de atributos que o tornem facilmente operável;
- Proteção contra erros do usuário: grau em que o sistema é capaz proteger os usuários de cometerem erros;
- Estética da interface: grau em que a interface de usuário possibilita a satisfação do usuário;

- Acessibilidade: grau em que o sistema é capaz de atender o maior leque de características e capacidades para atingir uma determinada meta em um determinado contexto de uso;

3.1.5 Confiabilidade

Grau em que um sistema, produto ou componente é capaz de executar corretamente um determinado conjunto de funções por um determinado intervalo de tempo sob determinadas condições de uso. Essa característica é atingida quando as subcaracterísticas são respeitadas:

- Maturidade: grau em que o sistema, produto ou componente atende as necessidades de confiabilidade sob condições normais de operação;
- Disponibilidade: grau em que o sistema, produto ou componente está operacional e acessível quando requisitado para uso;
- Tolerância a falhas: grau em que o sistema, produto ou componente é capaz de operar apesar de falhas de *hardware* ou *software*;
- Recuperabilidade: grau em que o produto, dada uma situação de falha, é capaz de recuperar o estado e os dados pré-falta;

3.1.6 Segurança da informação

Grau para o qual um produto ou sistema protege informações e dados para que pessoas ou outros produtos ou sistemas tenham o grau de acesso a dados adequado aos seus tipos e níveis de autorização. Esta característica é composta das seguintes sub-características:

- Confidencialidade: Grau para o qual um produto ou sistema garante que os dados sejam acessíveis somente àqueles autorizados a ter acesso;
- Integridade: Grau para o qual um sistema, produto ou componente impede o acesso não autorizado ou a modificação de programas de computador ou dados;

- Não repúdio: grau em que ações ou eventos podem ser provados como tendo ocorrido, de modo que os eventos ou ações não possam ser repudiados mais tarde;
- Prestação de contas: Grau para o qual as ações de uma entidade podem ser rastreadas exclusivamente para a entidade;
- Autenticidade: Grau em que a identidade de um sujeito ou recurso pode ser provada como sendo aquela reivindicada;

3.1.7 Manutenabilidade

Essa característica representa o grau de eficácia e eficiência com o qual um produto ou sistema pode ser modificado para melhorá-lo, corrigi-lo ou adaptá-lo a mudanças no ambiente e nos requisitos. Essa característica é composta das seguintes subcaracterísticas:

- Modularidade: Grau para o qual um sistema ou programa de computador é composto de componentes discretos, de modo que uma alteração em um componente tenha impacto mínimo em outros componentes;
- Reutilização: Grau em que um ativo pode ser usado em mais de um sistema ou na criação de outros ativos;
- Analisabilidade: Grau de eficácia e eficiência com o qual é possível avaliar o impacto em um produto ou sistema de uma mudança pretendida para uma ou mais de suas partes, ou para diagnosticar um produto por deficiências ou causas de falhas, ou para identificar peças a serem modificadas;
- Modificabilidade: Grau para o qual um produto ou sistema pode ser efetivamente e eficientemente modificado sem introduzir defeitos ou degradar a qualidade do produto existente;
- Testabilidade: Grau de eficácia e eficiência com o qual critérios de teste podem ser estabelecidos para um sistema, produto ou componente e testes podem ser realizados para determinar se esses critérios foram atendidos;

3.1.8 Portabilidade

Grau de eficácia e eficiência com o qual um sistema, produto ou componente pode ser transferido de um hardware, software ou outro ambiente operacional ou de uso para outro. Esta característica é composta das seguintes sub-características:

- Adaptabilidade: Grau para o qual um produto ou sistema pode ser efetivamente e eficientemente adaptado para hardware, software ou outros ambientes operacionais ou de uso diferentes ou em evolução;
- Instalabilidade: Grau de eficácia e eficiência com o qual um produto ou sistema pode ser instalado e/ou desinstalado com êxito em um ambiente especificado;
- Substituição: Grau para o qual um produto pode substituir outro produto de software especificado para o mesmo propósito no mesmo ambiente;

A partir desse modelo genérico de qualidade de software, são feitas as especificações de quais itens relevantes para que a qualidade, como nível de satisfação do cliente, seja respeitada.

3.2 Avaliação de Qualidade em sistemas embarcados

Em contrapartida a situação da definição do modelo de qualidade para produtos de software, não existe a definição de órgãos internacionais acerca do modelo de qualidade para projetos de sistemas embarcados.

Na prática, os modelos são definidos pelas empresas fabricantes e revisados internamente até que seja atingida a qualidade do produto ao consumidor. Nesse quesito, as empresas do ramo automobilístico são as responsáveis pela elaboração e execução dos modelos de qualidade.

Como parte das atividades do estágio, foi solicitada a criação de um modelo de qualidade de sistemas embarcados para o projeto no qual o estagiário foi alocado. A opção feita foi desen-

volver um modelo a partir do modelo de software realizando a adição, remoção e/ou adaptação dos campos do modelo de *software* de acordo com os requisitos de projeto negociados com o cliente.

A partir dos requisitos e da ISO 25010, foi proposto como modelo de qualidade de sistemas embarcados o modelo da Figura 3.2.

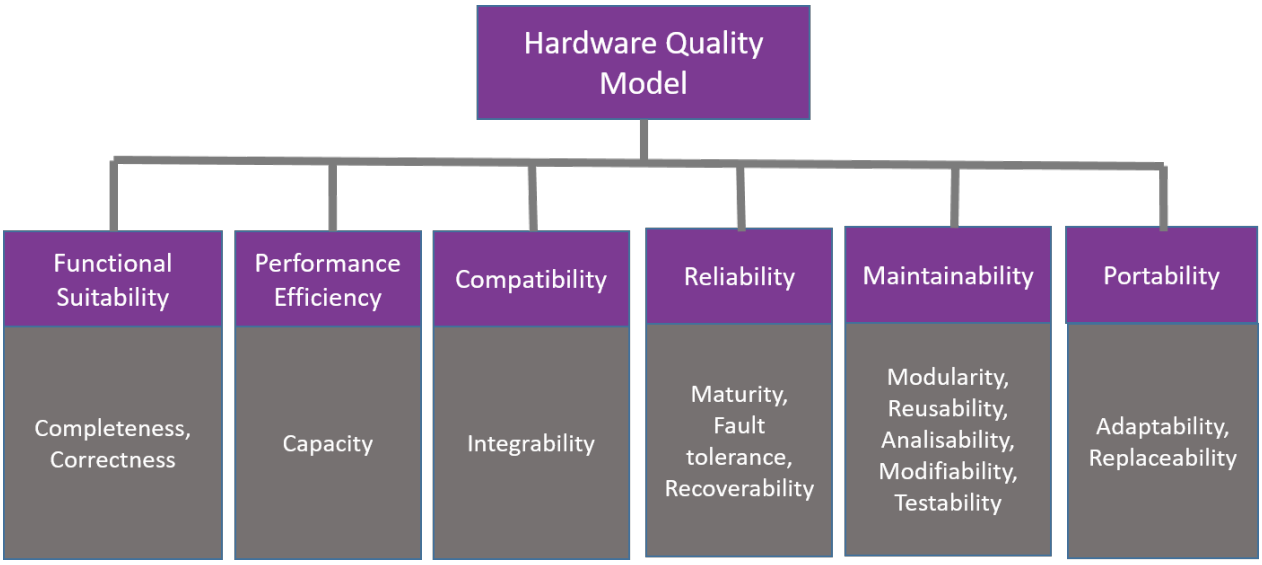

Figura 3.2: Modelo de qualidade de sistemas embarcados criado a partir da descrição da ISO 25010

4 Etapas da implantação do modelo de qualidade

Uma vez que o modelo de qualidade é definido, entra em cena outra parte relacionada ao planejamento das ações que vão garantir que as características do modelo de qualidade serão atendidas. Essa garantia vem a partir da realização de uma série de testes executados sobre a entidade a ser entregue ao cliente.

O processo de testes compreende o momento do planejamento, por meio definição das metas e estratégias de teste, definição dos casos de teste e execução. Serão detalhados nesse capítulo cada item desse processo.

4.1 Planejamento dos testes

Na fase de planejamento, o primeiro item a ser criado é o MTP (*Master Test Plan*). Esse documento deve conter todas as informações referentes aos objetivos dos testes, estratégias e técnicas empregadas. A estrutura e conteúdo do MTP são sugeridos pela norma *IEEE 829*.

A partir do documento adotado como referência, foi proposta uma estrutura para o MTP dividida em: Introdução, Histórico de edição, Metas da qualidade, Características de qualidade, Processo de qualidade, Estratégias de teste, Técnicas de teste, Planos de teste, Gestão de Riscos, Definição das métricas, Ferramentas, Canais de comunicação e Definição dos papéis dos membros da equipe.

4.1.1 Introdução

Essa seção deve descrever a finalidade e o escopo do documento.

4.1.2 Histórico de Edição

Essa seção contém um histórico de edições do MTP, que é, necessariamente, reeditado continuamente durante o ciclo de vida do projeto, devido a alterações no processo de desenvolvimento ou revisão de decisões tomadas anteriormente;

4.1.3 Metas de qualidade

Essa seção descreve os critérios de avaliação de maturidade dos entregáveis do projeto bem como os critérios de aceitação de cada uma das fases do desenvolvimento.

Para o projeto acompanhado durante o período do estágio, foi realizada a divisão em três fases. A primeira, corresponde a fase de concepção: Período em que o desenvolvimento é feito vislumbrando a construção de uma prova de conceito do sistema, ou seja, um artefato que é

capaz de demonstrar a viabilidade da solução do problema a partir da aplicação de um determinado tipo de técnica. A segunda fase corresponde à construção da solução propriamente dita, sendo caracterizada pelo desenvolvimento dos módulos do sistema e integração dos componentes. A última fase se refere a etapa de validação funcional do sistema como um todo.

Para cada fase, foram definidos os critérios de aceitação para os graus de maturidade em cada fase:

- **Alpha**
  - Grau para builds prematuras (dev build) antes de qualquer atividade de teste na qual é esperado retrabalho e serve como indicador do status das atividades do desenvolvimento. O critério de aceitação dessa fase é a inexistência de *bugs* críticos e validação do objetivo do módulo.
- **Beta**
  - Grau para *builds* intermediárias na qual são submetidos ao conjunto de testes os módulos desenvolvidos de forma unitária com todos os *features* parcial ou totalmente desenvolvidos. O critério de aceitação total dessa fase é de pelo menos 90% dos casos de teste aprovados e a inexistência de bugs catastróficos. A aceitação com ressalva acontece quando uma porcentagem menor do que 90% dos casos de teste são aprovados e não são encontrados bugs críticos. A reprovação da *build* se dá quando forem encontrados bugs catastróficos ou altos e taxa de aprovação menor do que 90% nos casos de teste.
- ***Release Candidate***
  - Grau para *builds* maduras após aprovação no ciclo de testes Beta destinado aos testes de integração dos módulos componentes do sistema, sendo esperado um mínimo de retrabalho relacionado ao conserto de *bugs* encontrados na versão anterior. O critério de aceitação total dessa fase é de 95% de aprovação nos casos de teste e a inexistência de *bugs* de severidade média ou alta. Nesse caso, não se aplica a aprovação com ressalvas. Qualquer violação nas condições de aceitação total implicam na reprovação da *build*

- ***Final***

  Grau para builds maduras onde todo o sistema é submetido aos planos de teste e o nível de retrabalho se resume a correção de *bugs* relacionados ao processo de funcionamento do equipamento em condições de operação. O critério de aceitação dessa build é a aprovação em todos os casos de teste e a inexistência de *bugs* identificados em qualquer nível de severidade. A aprovação com ressalva é dada em caso de 95% de aprovação nos casos de teste e inexistência de *bugs* médios ou críticos. *Bugs* de qualquer natureza implicam na reprovação da *build*.

Em suma, o fluxo das fases do projeto e dos níveis de maturidade podem ser vistos na Figura 4.1:

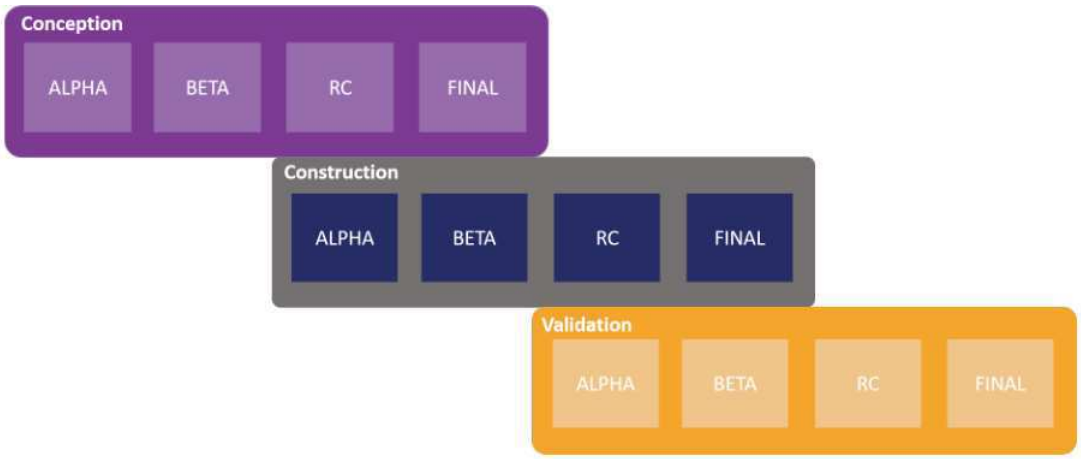

Figura 4.1: Fluxograma das etapas do desenvolvimento e do nível de maturidade das *builds*

4.1.4 Características de Qualidade

Essa seção descreve o modelo de qualidade adotado para o projeto em concordância com os itens selecionados da norma ISO 25010. Em linhas gerais, o procedimento descrito nessa seção corresponde ao que foi realizado na seção 3.2 desse documento.

4.1.5 Processos de qualidade

Essa seção descreve a definição do processo de criação dos planos de teste, partindo da especificação dos requisitos, desenvolvimento dos casos de teste, levantamento e análise dos riscos na execução dos casos de teste, estratégias de mitigação dos riscos identificados, levantamento dos equipamentos e arranjo físico do *setup* de teste necessário.

O fluxo de construção do plano e dos casos de teste podem ser observados nas Figuras 4.2 e 4.3.

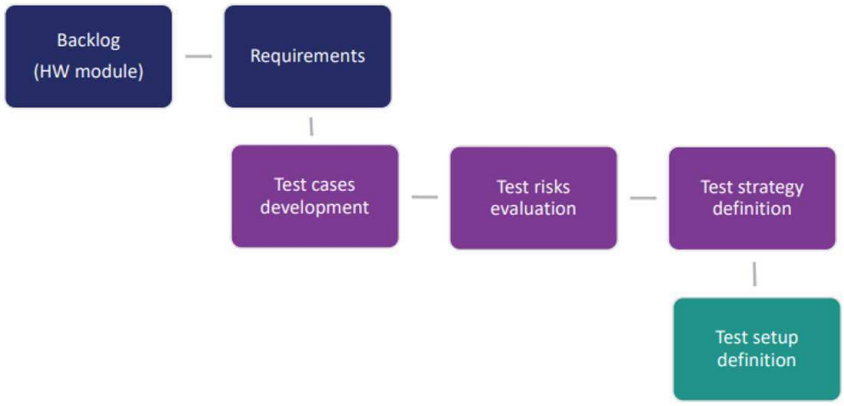

Figura 4.2: Fluxo de criação do plano de testes

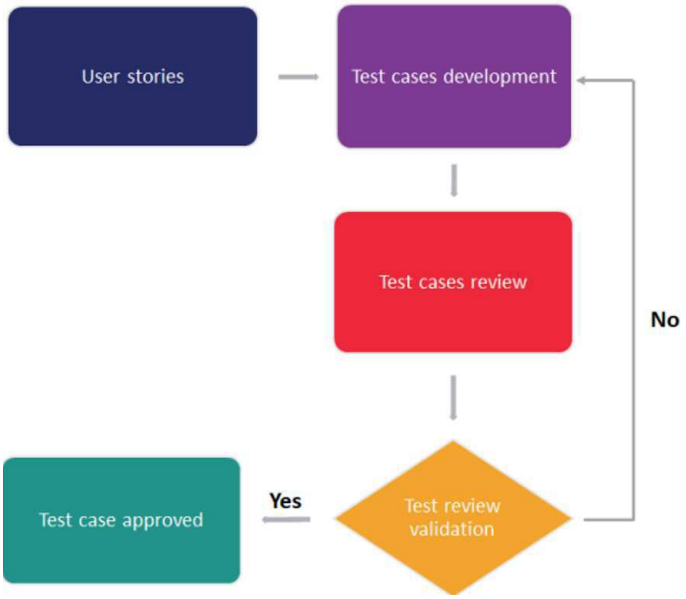

Figura 4.3: Fluxo de criação dos casos de teste incluindo o *loop* de reavaliação dos casos

4.1.6 Estratégias de testes

Essa seção descreve as estratégias adotadas para realização dos testes. As estratégias mais comuns nessa abordagem são os testes unitários, os testes funcionais manuais, os testes de tolerância a falhas e os testes de performance e *Stress* do sistema.

4.1.7 Análise do modo de falhas

No contexto de adaptação das estratégias de teste para projetos de Hardware, foi adotada a análise de efeitos em modo de falha, internacionalmente conhecida como *FMEA* (*Failure Mode and Effect Analysis*).

Os primeiros padrões para os documentos *FMEA* foram propostos pelas Forças Armadas dos Estados Unidos em 1949 (MIL-STD-1629A), sendo aplicados nas missões que levaram o homem a lua no final da década de 1960. Na década de 1980, lideradas pela Ford, as empresas do ramo automobilístico tornaram o FMEA parte obrigatória dos projetos, criando padrões como o SAE J1739.

O FMEA consiste na listagem dos Possíveis modos de falha, Efeitos decorrentes do modo de falha, Severidade da falha, Causa em potencial da falha observada, Probabilidade de ocorrência, Ações de controle, Detecção e Ações recomendadas. A descrição da elaboração desses itens pode ser observada em (5).

Finalmente, foi definido um modelo de tabela com os itens do FMEA de modo a ser preenchida de acordo com as demandas e componentes do projeto. Esse modelo pode ser observado nas Tabelas 1 e 2.

4.1.8 Técnicas de teste

Essa seção descreve quais são as técnicas de teste a serem usadas para validação do produto sob teste. Em geral, os testes são divididos em duas vertentes: a primeira se refere aos

Item	Identificador do módulo ou sistema
Função	Função do módulo ou sistema
Modo de falha potencial	Descrição do estado de erro
Severidade	Grau de impacto no funcionamento do sistema
Causa Potencial	Causa que levou o sistema ao estado de erro
Ocorrência	Nível de ocorrência do erro em situações de operação
Ações de controle (preventivas)	Listagem das ações, em termos de projeto, a serem aplicadas de modo a evitar o modo de falha
Detecção	Nível de facilidade na detecção do modo de falha
"Risk priority Number"(RPN)	Produto ponderado da severidade, Ocorrência e Detecção. Quanto maior o RPN maior é o nível de prioridade que o estado de falha deve ser posto para ser tratado
Ações recomendadas	Ações de mitigação para o estado de erro observado

Tabela 1: Modelo e descrição dos itens do FMEA

testes baseados em especificações e a outra em testes baseados na experiência.

Os testes baseados em especificações são direcionados a validação de campos, aplicação de valores limite (testes de fronteira), partições de equivalência, etc. Os testes baseados em experiência compreendem os desenvolvidos com base em situações ocorridas em experiências anteriores, de modo a complementar a cobertura dos testes baseados em especificações.

Para qualquer uma das técnicas empregadas os testes desenvolvidos podem ter natureza manual ou automática. Os testes manuais correspondem principalmente a verificação de fluxos e, no caso de projetos de *hardware*, a verificação de sinais elétricos a partir de equipamentos como multímetro e osciloscópio. Quando a natureza do teste é de repetição dos fluxos com mínimas variações, o teste pode ser automatizado de modo a otimizar o gasto de tempo para sua realização.

Severidade	
Extrema	10
	9
Alta	8
	7
Moderada	6
	5
	4
Baixa	3
	2
Nenhuma	1

Ocorrência		
Extrema	1 em 2	10
	1 em 3	9
Alta	1 em 8	8
	1 em 20	7
Moderada	1 em 80	6
	1 em 400	5
	1 em 2000	4
Baixa	1 em 15k	3
	1 em 150k	2
Nenhuma	1 em 1.5M	1

Detecção		
Muito Baixa	1 em 10	10
Baixa	1 em 20	9
	1 em 50	8
Moderada	1 em 100	7
	1 em 200	6
	1 em 500	5
Alta	1 em 1k	4
	1 em 2k	3
Extrema	1 em 5k	2
	1 em 10k	1

Tabela 2: Métricas de avaliação de severidade, ocorrência e detecção do FMEA

4.1.9 Planos de teste

Essa seção descreve o modelo de construção dos planos de teste. Essa descrição deve conter os campos de Título/descrição do teste, Identificação do sistema ou módulo, Objetivos, Precondições, Descrição dos passos da realização do teste, Resultados esperados, Critérios de aceitação, Número de execuções, Resultados obtidos, Status da execução (sucesso ou falha), Frequência de resultados positivos, Severidade e Observações acerca do teste.

4.1.10 Gestão de riscos

Essa seção descreve as ações a serem tomadas quando um risco na execução dos testes é identificado

4.1.11 Métricas de qualidade

Essa seção define quais são as métricas usadas na avaliação do módulo ou sistema quando submetido a avaliação de qualidade. Algumas métricas comumente empregadas são:

- Número de casos de teste por requisito;
- Número de casos de teste por status;
- Número de defeitos reportados;
- Número de defeitos abertos por severidade;
- Cobertura de testes unitários;
- Cobertura de testes de integração;
- Densidade de defeitos;

4.1.12 Ferramentas

Essa seção lista quais as ferramentas usadas pela equipe de qualidade na execução do plano. Para o caso de sistemas embarcados, também devem ser incluídos os equipamentos de medição presentes no conjunto de ferramentas necessárias à execução dos testes conforme descrito no plano.

4.1.13 Comunicação

Essa seção descreve quais são os meios de comunicação padrão usados para comunicação da equipe de qualidade com as demais equipes do projeto.

4.1.14 Equipe

Essa seção lista os membros da equipe de qualidade e quais são suas atribuições individuais dentro do escopo de qualidade definido no projeto.

4.2 Execução dos testes

Uma vez que o planejamento dos testes é concluído, o passo seguinte é a execução desse plano. Conforme o que foi especificado durante a fase de planejamento, a execução pode se dar de forma automática ou manual.

Os testes de natureza manual consistem na execução dos passos descritos no planejamento e comparação com os resultados esperados, de modo a validar o cenário de teste. Esse tipo de teste é empregado frequentemente em duas fases do projeto: Testes funcionais, destinados a validação das unidades funcionais de forma individual e testes de sistema ou integração, quando é necessária a validação da interação entre os módulos do sistema.

A vantagem do emprego dos testes manuais se dá pela inclusão do julgamento do testador na avaliação do fluxo analisado. Por outro lado, os testes manuais são excessivamente demorados, não são aplicáveis quando o numero de execuções é alto e estão mais sujeitos a resultados errôneos.

Os testes de natureza automática, por sua vez, consistem no emprego de rotinas de *software* para aplicação de estímulos e validação dos resultados. Esse tipo de teste é empregado principalmente na realização de testes de performance, regressão e stress.

A vantagem do emprego de testes automatizados se caracteriza pela possibilidade de repetição em larga escala em curtos intervalos de tempo quando comparada aos testes manuais. Sua desvantagem, no entanto, é o tempo necessário para desenvolvimento das rotinas e escolha do *framework* adotado. Além disso, os testes automáticos não incluem o julgamento do testador no decorrer do teste, que é um fator importante.

Durante o desenvolvimento das atividades do estágio, foi observado que cerca de 87% dos casos de teste de *hardware* elaborados tem natureza de execução manual. Esse índice era esperado, dado que a maioria dos testes executados e projetados para *hardware* envolvem a simulação de situações como interrupção da alimentação, desconexão de periféricos, etc.

Uma vez executado o ciclo de teste, deve ser gerado um relatório dos resultados do ciclo. Para qualquer teste, seja automático ou manual, a aprovação está sujeita aos critérios de aceitação definidos no plano. Em caso de falha em frente aos critérios de aceitação ou de identificação de *bugs*, estes devem ser reportados para serem colocados como parte das atividades do time de desenvolvimento. É importante lembrar que, a depender da quantidade de *bugs* ou falhas nos casos de teste, o artefato sob teste pode ser reprovado de acordo com os critérios de maturidade específicos do ciclo.

5 Adaptação do sistema de integração contínua

As metodologias ágeis vem se consolidando atualmente no que tange o desenvolvimento de *software*.Sua aplicação proporciona vantagens do ponto de vista gerencial, no tocante ao desenvolvimento dos entregáveis em etapas, e do ponto de vista de qualidade, tornando possível a aplicação dos diferentes tipos de teste durante cada etapa do desenvolvimento.

O fluxo de desenvolvimento, com a aplicação de algum tipo de metodologia ágil, começa com o planejamento das atividades (atomização dos módulos a serem desenvolvidos) sendo seguidas pelo desenvolvimento e fase de validação por meio da aplicação dos testes.

No desenvolvimento, os colaboradores desenvolvem os módulos e realizam, diariamente, a postagem dos códigos-fonte em um repositório remoto. Esses servidores armazenam todo o histórico de desenvolvimento do projeto como um todo, possibilitando a recuperação de versões anteriores dos módulos em caso de falha, além de ser um backup remoto do projeto.

Atrelado ao servidor de controle de versão está o servidor de integração contínua, que tem como função gerar o ambiente de compilação e gerar, a partir dos arquivos do repositório,

um artefato. Esse artefato pode ser um módulo, subsistema ou até mesmo o sistema completo, a depender da fase de desenvolvimento. Além disso, o servidor de integração contínua também pode ser configurado para realização de testes automáticos, de modo a validar um conjunto mínimo de características por meio da aplicação de testes de regressão.

Durante o desenvolvimento das atividades do estágio, foi proposta a adaptação do sistema de integração contínua para a vertente de *hardware* do projeto. Para que esse resultado fosse atingido, foi necessário dominar cada fase do processo, que é composto pela configuração da ferramenta de integração contínua, pela configuração do compilador, ferramenta de gravação e a criação da ferramenta de gerenciamento de testes automáticos.

5.1 Ferramentas de integração contínua

Devido à importância do processo de integração contínua no contexto de desenvolvimento ágil, várias ferramentas estão disponíveis no mercado como, por exemplo, o *Gitlab CI*, *Travis CI*, *Codeship* e *Jenkins*. Devido a larga utilização e característica *opensource*, a ferramenta escolhida foi o *Jenkins*.

Uma vez baixado a partir do site oficial, o serviço do *Jenkins* deve ser iniciado. Essa etapa é realizada por meio do comando executado via linha de comando na pasta na qual o download foi realizado:

```
java -jar jenkins.war
```

Uma vez instalado, o passo a seguir é configurar um novo *Job*, que corresponde ao nome do projeto a ser criado. Em seguida, devem ser fornecidas as credenciais e os endereços do servidor de controle de versão para que o *Jenkins* tenha acesso ao repositório, bem como deve ser informado qual *branch* deve ser importado.

O próximo passo se refere a configuração do ambiente de compilação e da sequência de *build* dos artefatos. Para essa etapa, foi utilizada uma ferramenta de *headless build* disponibilizada pelo *Atollic TrueStudio* para compilar, via linha de comando, projetos desenvolvidos em seu ambiente. Dado que o projeto a ser compilado foi desenvolvido no *Atollic*, não foi necessário realizar nenhuma configuração adicional. O *script* usado para realizar todo o processo de build pode ser observado a seguir.

```
echo off
:: set environment variables

::project dir is the folder downloaded from git
SET PROJECT_DIR="C:\Users\Projeto XX\.jenkins\workspace\teste-zero"

::ATOLLIC_DIR is the path where the the headless build tool is located
SET ATOLLIC_DIR="C:\Program Files (x86)\Atollic\TrueSTUDIO for STM32 9.1.0\ide"

::CHECKOUT dir is any folder you want
SET CHECKOUT_DIR="C:\Jenkins\git-downloaded\buildWS"

::step 1: cd to atollic dir
cd %ATOLLIC_DIR%

::step 2: call headless builder
call headless.bat -data %CHECKOUT_DIR% -import %PROJECT_DIR% -cleanBuild all
set BUILD_STATUS=%ERRORLEVEL%
if not %BUILD_STATUS%==0 (exit /b 1)
```

Uma vez que o processo de compilação é finalizado corretamente, é gerado um arquivo binário (*.elf) que deve ser gravado no microcontrolador. Essa etapa é realizada pelo OpenOCD, que é uma ferramenta de *Debug*, mas que também pode ser usada para esse propósito. O trecho do *script* responsável por essa parte pode ser observada a seguir:

```
::copy .elf to ocd path
cd %PROJECT_DIR%\Debug
copy *.elf %OCD_PATH%
cd C:\openocd\bin
call openocd -f board\st_nucleo_f7.cfg -c "program stm32-nucleo144-f7.elf verify reset exit"
```

Ao fim dessa etapa, o artefato já foi construído e gravado no microcontrolador, estando pronto para a inicialização dos testes.

5.2 Gerenciador de testes

A última etapa do processo de integração contínua se dá a partir da execução de uma sequência de testes automáticos sob o artefato que foi construído nas etapas anteriores. Ao contrário do que é realizado com artefatos de *software*, o teste automático do *software* embarcado é feito, quase que via de regra, por um outro embarcado, devido a necessidade da geração e leitura de sinais de resposta por parte do dispositivo sob teste.

A primeira fase do desenvolvimento da solução corresponde a coleta dos requisitos de projeto. Para esse caso foram coletados os seguintes requisitos:

- A solução deve ser capaz de carregar um arquivo com a descrição dos testes a serem realizados. Esse arquivo deve conter informações sobre o periférico a ser testado, quais os pinos usados no teste, a quantidade de execuções do teste e os estímulos a serem gerados ou lidos durante a execução dos testes. A descrição desse arquivo deve ser realizada tanto para o dispositivo sob teste (DUT -*Device under test*) quanto para o testador (*Tester Device*);
- A solução deve ser capaz de se comunicar, via serial, com o dispositivo sob teste e o testador de forma assíncrona e bloqueante, sendo qualquer evento disparado pelo gerenciador de execução dos testes funcionando na máquina em que os testes estão sendo executados.
- A solução deve gerar *logs* de todas as operações de teste realizadas, bem como o *status* (falha ou sucesso) de cada uma das etapas assim como o *status* geral da execução do teste carregado do arquivo.

A partir dos requisitos, foi proposta a arquitetura para a versão do gerenciador de testes no PC observada na Figura 5.1 e a versão a ser implementada na versão do embarcado na Figura 5.2.

Figura 5.1: Arquitetura proposta para a versão do gerenciador de testes no PC

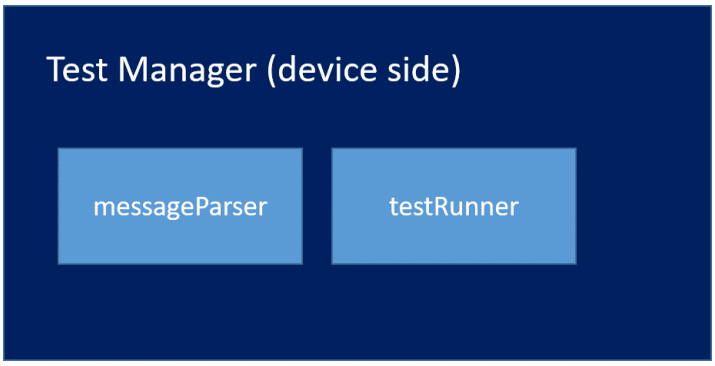

Figura 5.2: Arquitetura proposta para a versão do gerenciador de testes para o embarcado

A parte a ser implementada no lado do PC é responsável pela criação da conexões via serial (*serialManip*), pela importação da descrição do teste (*jsonManip*) e geração das mensagens a serem enviadas pela serial com o passo do teste, pela quebra das mensagens de resposta das mensagens recebidas do lado do embarcado (*messageParser*) e pela geração dos relatórios de execução (*reportGenerator*).

A versão implementada no lado do embarcado, foi desenvolvida na forma de biblioteca, de forma que possa ser adicionada a outros projetos. Um dos problemas que ela apresenta é que é totalmente dependente do *hardware* no qual o projeto está sendo desenvolvido.

Para a descrição dos testes foi desenvolvido um modelo, escrito em formato JSON (*Ja-*

vaScript Object Notation), com as características do teste a ser executado. Nessa descrição, devem estar contidas as informações sobre o dispositivo que está sendo testado, sobre o periférico, sobre o tipo de teste, quais as portas em que os testes serão realizados no dispositivo sob teste e no testador e quais os estímulos a ser usados no teste. A seguir, um exemplo de um modelo de descrição de teste:

```
"deviceId": "STM32",
"peripheralName": "GPIO",
"testType": "Read",
"dutPins": [
                {"GPIO": "GPIOG", "GPIO_PIN": "GPIO_PIN_0"},
                ],
"tsmPins": [
                {"GPIO": "GPIOG", "GPIO_PIN": "GPIO_PIN_1"},
                ],
"stimulus": [
        "HIGH", "LOW", "HIGH", "LOW", "HIGH", "LOW", "HIGH",
        "LOW", "HIGH", "LOW", "HIGH", "LOW", "HIGH", "LOW"]

"executionTimes": "30"
```

Para esse exemplo, o teste é realizado em uma *STM32*, no periférico *GPIO*, realizando operação de leitura no pino *G0* conforme a sequência definida no campo *stimulus* com repetição de trinta vezes. O campo *tsmPins* determina qual pino é responsável pela geração do sinal de estímulo, havendo a necessidade de estar conectado fisicamente a porta na qual está sendo realizada a operação de leitura.

Uma vez que o teste é carregado, o gerenciador de teste gera a mensagem a ser enviada ao embarcado com as características do passo a ser executado. Essa mensagem tem tamanho fixo de dez *bytes* e é codificada da seguinte forma:

- byte[0]: Periférico a ser testado;
- byte[1]: Tipo do teste a ser realizado;
- byte[2-9]: Informação adicional acerca do periférico;

Para cada tipo de teste realizado foi associado uma codificação. Para o teste de leitura de *GPIO* descrito no exemplo anterior, a mensagem gerada para o dispositivo sob teste foi 'grgg1xxxxx', na qual:

- g: periférico sob teste (GPIO);
- r: tipo do teste(READ);
- g: informação adicional sobre o perifério (GPIO);
- g: primeira parte da identificação da porta (GPIOG);
- 1: segunda parte da identificação da porta (GPIO_PIN_1);
- x: informação irrelevante para o teste;

Em contrapartida, a mensagem gerada para o testador é 'gwgg1hxxxx':

- g: periférico sob teste (GPIO);
- w: tipo do teste(WRITE);
- g: informação adicional sobre o perifério (GPIO);
- g: primeira parte da identificação da porta (GPIOG);
- 0: segunda parte da identificação da porta (GPIO_PIN_0);
- h: sinal a ser colocado no pino (h = HIGH, l = LOW);

Uma vez que as mensagens estão montadas, o pacote é enviado para o dispositivo sob teste, que faz a separação da mensagem, executa a operação e devolve uma mensagem, sinalizando que a operação foi executada ao Gerenciador. Em seguida, é enviada a mensagem ao dispositivo de teste, que executa o passo e devolve o resultado. Finalmente, os resultados do teste são avaliados e registrados no *log* de atividades, até que sejam completadas as execuções especificadas no teste.

6 Resultados

Devido ao final do tempo previsto para conclusão do estágio, o Gerenciador de teste não foi integrado à pilha dos processos de integração contínua, sendo necessários ajustes na interface do gerenciador para que este seja incluído no *script* de integração contínua.

Com relação ao processo de qualidade, a construção da documentação do processo já consiste no alcance dos resultados previstos.

Como trabalhos futuros, o gerenciador de testes pode ser mais generalista, com a criação de uma camada de abstração de *hardware*, além de uma interface para geração dos testes em alto nível.

7 Considerações Finais

O desenvolvimento das atividades durante o período do estágio foi altamente proveitoso para compreensão do processo e das técnicas de validação de *software* conforme padronização internacional, possibilitando o vislumbre de uma área totalmente nova e extremamente importante na esfera profissional.

Além disso, a oportunidade de participar do desenvolvimento de um processo equivalente para validação de *hardware* embarcado, a adaptação para um sistema de integração contínua e o desenvolvimento de ferramentas de validação foi altamente construtiva em uma área de formação que não é comum ao estudante de engenharia elétrica.

8 Bibliografia

[1] YOUNG, Michal. *Software Testing and Analysis: Procedures and Techniques*. *John Wiley Sons. 2008*

[2] TAGUE, Nancy. *The Quality Toolbox* . 2nd ed. *ASQ Quality Press. 2005*

[3] BERGERON, Janick. *Writing Testbenches: Functional Verification of HDL models* . 2nd Ed. *Kluwer Academic Publishers. 2003*

[4] CHEMTURI, Murali. *Software Quality Assurance: Best Practices, Tools and Techniques for Software Developers. J.Ross Publishing. 2010*

[5] CARLSON, Carl S. *Understanding and Applying the Fundamentals of FMEAs 2015 Annual Reliability and Maintainability Symposium. 2015*

[6] INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. *IEEE 829-1998. Software Test Documentation. 1998*

[7] INTERNATIONAL ORGANIZATION OF STANDARDIZATION (ISO). *ISO/IEC 25010:2011. Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.*