UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE MECÂNICA
CURSO DE GESTÃO DO DESENVOLVIMENTO DE PRODUTOS

GEOVANNI DIAZ DE ARAÚJO

METODOLOGIA E PROCESSOS DE VALIDAÇÃO DE SOFTWARE EMBARCADO

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2016

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE MECÂNICA
CURSO DE GESTÃO DO DESENVOLVIMENTO DE PRODUTOS

GEOVANNI DIAZ DE ARAÚJO

METODOLOGIA E PROCESSOS DE VALIDAÇÃO DE SOFTWARE EMBARCADO

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2016

GEOVANNI DIAZ DE ARAÚJO

METODOLOGIA E PROCESSOS DE VALIDAÇÃO DE SOFTWARE EMBARCADO

Trabalho de Conclusão de Curso apresentado ao Curso de Gestão do Desenvolvimento de Produtos da Universidade Tecnológica Federal do Paraná, Campus Curitiba, como requisito parcial à obtenção do título de Especialista em Gestão do Desenvolvimento de Produtos.

Orientador: Prof. Nilton Luiz Cararo

CURITIBA

2016

.

Araujo, Geovanni
Metodologia e Processos de Validação de Software Embarcado / Geovanni Araujo.
Curitiba. UTFPR, 2016
57 f. : il. ; 30 cm

Orientador: Prof. Nilton Luiz Cararo
Monografia (Trabalho de Conclusão de Curso) - Universidade Tecnológica Federal do Paraná. Curso de Gestão do Desenvolvimento de Produtos. Curitiba, 2016.
Bibliografia: f. 45 – 46

1. Validação. 2. Software. I. Cararo, Nilton, orient. II. Universidade Tecnológica Federal do Paraná. Curso de Gestão do Desenvolvimento de Produtos. III. Metodologia e Processos de Validação de Software Embarcado.

CDD: 630

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba
Departamento Acadêmico de Mecânica
Curso de Gestão do Desenvolvimento de Produtos

TERMO DE APROVAÇÃO

Trabalho de Conclusão de Curso - TCC

METODOLOGIA E PROCESSOS DE VALIDAÇÃO DE SOFTWARE EMBARCADO

por

GEOVANNI DIAZ DE ARAÚJO

Monografia apresentada às ___ horas ___ min. do dia ___ de ___ de 2016 como requisito parcial para obtenção do título de ESPECIALISTA EM GESTÃO DO DESENVOLVIMENTO DE PRODUTOS, Curso de Gestão do Desenvolvimento de Produtos da Universidade Tecnológica Federal do Paraná, Campus Curitiba. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo-assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Banca examinadora:

Prof. Dr. XXX
UTFPR

Prof. Dr. WWW
UTFPR

Prof. Dr. XXX
UFPR

Prof. NILTON CARARO
UTFPR
Orientador

A “Ata de Defesa” e o decorrente “Termo de Aprovação” encontram-se assinados e devidamente depositados na Coordenação do Curso de Gestão do Desenvolvimento de Produtos da UTFPR Campus Curitiba-PR, conforme Norma aprovada pelo Colegiado de Curso.

Agradeço a Deus, pois sem ele eu não teria forças para essa jornada, agradeço a meus professores e aos meus colegas que me ajudaram na conclusão da monografia.

AGRADECIMENTOS

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender. A palavra mestre, nunca fará justiça aos professores dedicados aos quais sem nominar terão os meus eternos agradecimentos.

.

Procure ser um homem de valor, em vez de ser um homem de sucesso.

RESUMO

ARAÚJO, Geovanni. Metodologia e Processos de Validação de Software Embarcado. 57 f. TCC (Curso de Gestão do Desenvolvimento de Produtos), Universidade Tecnológica Federal do Paraná. Curitiba, 2016.

O objetivo deste estudo é propor um método científico e qualitativo, mais refinado, de referência e de boas práticas em validação de software embarcado baseado em métodos e processos de testes de software, a partir de literatura existente, que mostra e prova a importância de validação de sistemas no desenvolvimento de novos produtos com programa embarcado do segmento automotivo, sobretudo, no que diz respeito à redução de custos em projetos e também agregar informações importantes às etapas de testes no desenvolvimento de produto. Será explorado o uso de hardwares (dispositivos de testes), ferramentas de teste de software (Simuladores, Emuladores e etc) e ainda técnicas (Casos de Teste e verificação de cobertura de teste) para Testes de Módulo e de Integração (ambos Testes Caixa Branca) e Testes de Sistema ou Funcionais em bancada e no veículo (ambos Testes Caixa Preta).

Palavras-chave: Validação. Software. Hardware.

ABSTRACT

ARAÚJO, Geovanni. Methodology and Processes of Embedded Software Validation. 57 f. TCC (Course of Management of the Products Development) - Federal University of Technology - Paraná. Curitiba, 2016.

The aim of this study is to propose a scientific and qualitative method, more refined, of reference and best practices in embedded system validation based on methods and software testing processes from the existing literature, that shows and proves the importance of the software validation in the development of new automotive industry products that have embedded system in its composition, especially with respect to project cost reduction and also add relevant information to this stage of development. It will be explored the use of hardware (devices and test equipment), development and software testing tools (Compilers, Debuggers, Simulators, Emulators, plates and Evaluation Prototypes) and also testing techniques (Test Case and coverage verification of testing based in Project Requirements) for module and integration tests (both white box tests) and system or functional testing, at bench and also in the vehicle (black box tests).

Keywords: Validation. Software. Hardware.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS E ACRÔNIMOS

HW	Hardware
SW	Software

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de software é uma atividade de crescente importância na sociedade contemporânea. A utilização de microcomputadores e microcontroladores nas mais diversas áreas do conhecimento humano tem gerado uma crescente demanda por soluções computadorizadas (com grande destaque para os sistemas embarcados), com alto grau de controle, lógicas de controle e acionamentos inteligentes e que gerem sistemas robustos e seguros.

O software de sistema embarcado é diferente do programa de aplicação (executado em um microcomputador) devido à forte interconexão com periféricos, às restrições impostas pela aplicação e ao modo como é desenvolvido. O SW de um sistema embarcado acessa diretamente o hardware e deverá ser escrito de forma a se adaptar aos recursos de hardware disponíveis (KANG; KWON, 2005).

De maneira geral, sistemas embarcados são dispositivos capazes de desempenhar tarefas complexas. O software (ou em um contexto mais amplo – o sistema) embarcado tem se tornado complexo e quando se associa tal complexidade a grandes volumes de produção e impossibilidade de atualizações futuras, percebe-se a importância de testes no desenvolvimento do mesmo. Estima-se que 80% das falhas em um sistema embarcado são causadas por falhas na lógica de programação e não por hardware, mesmo o programa representando entre 10% e 20% do sistema (SEO, 2008). O custo de um defeito grave de software, se descoberto em campo, pode gerar prejuízos. Essas razões estão obrigando fabricantes de sistemas embarcados a executar testes de código de forma sistemática. Os Testes de SW são fundamentais em todos os ramos da engenharia, sobretudo, nas áreas militar, biomédica, automotiva e aeroespacial, onde erros não são aceitos e podem ser fatais.

O objetivo deste estudo é propor um método científico e qualitativo, mais refinado, de referência e de boas práticas em validação de sistema embarcado baseado em métodos e processos de testes de software, a partir de literatura existente, que mostra e prova a importância de validação de software no desenvolvimento de novos produtos do segmento automotivo, sobretudo, no que diz respeito à redução de custos em projetos e também agregar informações

importantes às etapas de testes no desenvolvimento de produto. Será explorado o uso de hardwares (dispositivos de testes), ferramentas de teste de software (Simuladores, Emuladores e etc) e ainda técnicas (Casos de Teste e verificação de cobertura de teste) para Testes de Módulo e de Integração (ambos Testes Caixa Branca) e Testes de Sistema ou Funcionais em bancada e no veiculo (Testes Caixa Preta).

2 ENGENHARIA DE SOFTWARE

A Engenharia de Software surgiu para melhorar a qualidade dos produtos com código embarcado e aumentar a produtividade no processo de desenvolvimento. Ela trata de aspectos relacionados ao estabelecimento de processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento de programas.

Em uma abordagem de engenharia de SW, inicialmente o problema a ser tratado deve ser analisado e decomposto em partes menores. Neste cenário, pessoas têm de trabalhar em equipes, o esforço tem de ser planejado, coordenado e acompanhado e a qualidade do que se está produzindo tem de ser sistematicamente avaliada (PFLEEGER, 2004).

2.1 QUALIDADE DE SOFTWARE

A qualidade de um código é um conceito com múltiplas perspectivas (de usuário, de desenvolvedor e do cliente) e que envolve diferentes características (testabilidade, corretitude, usabilidade, confiabilidade, eficiência, manutenibilidade, portabilidade e segurança) que devem ser alcançadas em níveis diferentes, dependendo do propósito do programa (PFLEEGER, 2004).

Abordagens de qualidade de processo, tal como a série de padrões ISO 9000, sugerem que melhorando a qualidade do processo de desenvolvimento de SW é possível melhorar a qualidade dos produtos (SW) resultantes. Um processo de desenvolvimento de SW envolve atividades que podem ser classificadas em:

- Atividades de Desenvolvimento: são as que contribuem diretamente para o desenvolvimento do produto a ser entregue ao cliente, tais como especificação e análise de requisitos, projeto e implementação.
- Atividades de Gerência: são aquelas relacionadas ao planejamento e acompanhamento gerencial do projeto.

- Atividades de Garantia da Qualidade: são aquelas relacionadas com a garantia da qualidade do produto e do processo utilizado, tais como revisões e inspeções de produtos do desenvolvimento.

2.2 PROCESSO DE SOFTWARE

Um processo de desenvolvimento de software é o conjunto de atividades, métodos, práticas e transformações que guiam pessoas na produção de programas embarcados. Um processo eficaz deve, claramente, considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os procedimentos necessários e a habilidade, o treinamento e a motivação do pessoal envolvido (PRESSMAN, 2002).

Os elementos, de maneira hierárquica, que compõem um processo de software, são (PFLEEGER, 2004):

Processo de Software
Processos
Atividades
Pré-atividades
Subatividades (atividades secundárias)
Artefatos
Insumos
Produtos
Recursos
Recursos Humanos
Ferramentas de Software
Hardware
Procedimentos
Métodos
Técnicas
Roteiros

Na base, sobre a qual o processo de desenvolvimento deve ser construído, estão as atividades-chave do processo de desenvolvimento de SW:

análise e especificação de requisitos, projeto, implementação e testes. Porém, a definição de um processo envolve a escolha de um modelo de ciclo de vida, o detalhamento de suas macro atividades, a escolha de métodos, técnicas e procedimentos e a definição de recursos e artefatos.

Um processo de desenvolvimento de software não pode ser definido de forma universal. Para ser eficaz e conduzir à construção de produtos de boa qualidade, o processo deve ser adequado ao domínio da aplicação e ao projeto específico. Deste modo, processos devem ser definidos caso a caso, considerando-se as especificidades da aplicação, a tecnologia a ser adotada na sua construção, a organização onde o produto será desenvolvido e o grupo de desenvolvimento (PRESSMAN, 2002).

O objetivo de se definir um processo é favorecer a produção de sistemas de alta qualidade, atingindo as necessidades dos usuários finais, dentro de um cronograma e um orçamento previsíveis.

A escolha de um modelo de ciclo de vida (ou modelo de processo) é o inicio para a definição de um processo de desenvolvimento de software. Um modelo de ciclo de vida organiza as macro atividades básicas, estabelecendo precedência e dependência entre as mesmas (SOMMERVILLE, 2003).

Um modelo de ciclo de vida pode ser entendido como as atividades que devem ser executadas durante um projeto. Para a definição completa do processo, a cada atividade, devem ser associadas técnicas, ferramentas e critérios de qualidade, entre outros, formando uma base sólida para o desenvolvimento. Adicionalmente, outras atividades, tipicamente de cunho gerencial, devem ser definidas, entre elas atividade de gerência, de controle e de garantia da qualidade (PFLEEGER, 2004).

Sendo assim, a razão para ser adotado um modelo de processo é, essencialmente, para ajudar na organização de um projeto, uma vez que, com um modelo as atividades, a serem executadas ao longo do projeto, são bem definidas. O ciclo de vida de um programa envolve as seguintes fases (PRESSMAN, 2002):

- *Planejamento:* o objetivo do planejamento de projeto é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos. Uma vez estabelecido o escopo de

software, uma proposta de desenvolvimento deve ser elaborada, isto é, um plano de projeto deve ser elaborado configurando o processo a ser utilizado no desenvolvimento de software. À medida que o projeto progride, o planejamento deve ser detalhado e atualizado regularmente. Pelo menos ao final de cada uma das fases do desenvolvimento (análise e especificação de requisitos, projeto, implementação e testes), o planejamento como um todo deve ser revisto e o planejamento da etapa seguinte deve ser detalhado. O planejamento e o acompanhamento do progresso fazem parte do processo de gerência de projeto;

- *Análise e Especificação de Requisitos:* nesta fase, o processo de levantamento de requisitos é intensificado. O escopo deve ser refinado e os requisitos identificados. Para entender a natureza do software a ser construído, o engenheiro de software tem de compreender o domínio do problema, bem como a funcionalidade e o comportamento esperados. Uma vez identificados os requisitos do sistema a ser desenvolvido, estes devem ser modelados, avaliados e documentados. Uma parte vital desta fase é a construção de um modelo descrevendo o que o software tem de fazer (e não como fazê-lo);
- *Projeto:* esta fase é responsável por incorporar requisitos tecnológicos aos requisitos essenciais do sistema, modelados na fase anterior e, portanto, requer que a plataforma de implementação seja conhecida. Basicamente, envolve duas grandes etapas: projeto da arquitetura do sistema e projeto detalhado. O objetivo da primeira etapa é definir a arquitetura geral do software, tendo por base o modelo construído na fase de análise de requisitos. Esta arquitetura deve descrever a estrutura de nível mais alto da aplicação e identificar seus principais componentes. O propósito do projeto detalhado é detalhar o projeto do software para cada componente identificado na etapa anterior. Os componentes de

software devem ser sucessivamente refinados em níveis de maior detalhamento, até que possam ser codificados e testados;

- *Implementação:* o projeto deve ser traduzido para uma forma passível de execução pela máquina. A fase de implementação realiza esta tarefa, isto é, cada unidade de software do projeto detalhado é implementada;
- *Testes:* inclui diversos níveis de testes, a saber, teste de unidade, teste de integração e teste de sistema. Inicialmente, cada unidade de software implementada deve ser testada e os resultados documentados. A seguir, os diversos componentes devem ser integrados sucessivamente até se obter o sistema. Finalmente, o sistema como um todo deve ser testado;
- *Entrega e Implantação:* uma vez testado, o software deve ser colocado em produção. Para tal, contudo, é necessário treinar os usuários, configurar o ambiente de produção e, muitas vezes, converter bases de dados. O propósito desta fase é estabelecer que o software satisfaz os requisitos dos usuários. Isto é feito instalando o software e conduzindo testes de aceitação (validação). Quando o software tiver demonstrado prover as capacidades requeridas, ele pode ser aceito e a operação iniciada.
- *Operação*: nesta fase, o sistema é utilizado pelos usuários no ambiente de produção;
- *Manutenção:* o programa sempre sofrerá mudanças após ter sido entregue para o usuário. Alterações ocorrerão porque erros foram encontrados, porque o software precisa ser adaptado para acomodar mudanças em seu ambiente externo, ou porque o cliente necessita de funcionalidade adicional ou aumento de desempenho. Muitas vezes, dependendo do tipo e porte da manutenção necessária, essa fase pode requerer a definição de um novo processo, onde cada uma das fases precedentes é reaplicada no contexto de um software existente ao invés de um novo.

Ao longo deste trabalho as fases Implementação, Testes, Entrega e Implantação, Operação e Manutenção serão bastante mencionadas. Algumas dessas fases terão uma abordagem mais aprofundada do que outras. Mas o foco principal desta pesquisa é, sem duvida, a fase de Testes.

3 ESTUDO TEÓRICO DE MÉTODOS E PROCESSOS DE VALIDAÇÃO DE SISTEMAS EMBARCADOS

Considerando que boa parte do custo de desenvolvimento de sistemas embarcados é devido a testes e correções de bugs (termo técnico para erros ou falhas de software), uma ferramenta de testes ou metodologia de desenvolvimento que não seja financeiramente viável, não terá aplicação real para grande parte das empresas (PFALLER, 2006). Assim, o teste de sistema embarcado é uma importante área de pesquisa, onde são buscadas técnicas de teste que maximizem o número de falhas encontradas ainda em tempo de projeto e a um custo satisfatório. Muitas das soluções pesquisadas envolvem aspectos relativos ao teste e também ao projeto do produto desde a sua concepção.

3.1 MODELOS DE DESENVOLVIMENTO DE SOFTWARE

Na área de software existem modelos de desenvolvimento, que de maneira geral seriam os antecessores das diversas metodologias para desenvolvimento de projeto existentes e aplicadas atualmente. Os principais modelos de desenvolvimento de SW são (PRESSMAN, 2002):

1. *Modelo em Cascata:* é um dos mais tradicionais por parecer ser mais simples e organizado, porém durante o desenvolvimento do projeto pode ocorrer inúmeras falhas decorrentes deste modelo. Nele as atividades do processo de desenvolvimento são estruturadas, como o próprio nome já diz, em cascata, onde a saída de uma etapa é a entrada para a próxima. O modelo é criticado por ser linear, rígido e monolítico. Argumenta que cada atividade apenas deve ser iniciada quando a outra estiver terminada e verificada. E é considerado monolítico por não introduzir a participação de clientes e usuários durante as atividades do desenvolvimento, apenas o código implementado e entregue. Não existe a possibilidade do cliente verificar antecipadamente o produto

para detectar eventuais problemas. Outras desvantagens do modelo em Cascata são:

- Não fornece feedback entre as fases e não permite atualização ou redefinição das etapas anteriores.
- Não suporta modificações nos requisitos.
- Não prevê a manutenção.
- Não permite a reutilização.
- É excessivamente sincronizado.
- Se ocorrer um atraso todo o processo é afetado.
- Demora muito para ser entregue o software.

2. *Modelo Iterativo:* é uma estratégia de planejamento de retrabalho em que o tempo de revisão e melhorias de partes do sistema são pré-definidos. Uma diferença típica é que a saída de um incremento não é necessariamente assunto de um refinamento futuro, e seu teste ou retorno do usuário não é utilizado como entrada para planos de revisão ou especificações para incrementos sucessivos. A ideia primordial é desenvolver um sistema de software incremental, permitindo ao desenvolvedor tirar vantagem daquilo que foi aprendido durante a fase inicial de desenvolvimento. As desvantagens do modelo são:
   - Durante o desenvolvimento é necessário adaptar e refinar o sistema, assim, pode ser que no final se tenha um resultado bem diferente da ideia original.
   - Pode acontecer a continuação do sistema e aparição de muitos requisitos novos.
   - Gerentes, acostumados com a forma linear, podem ter problemas na hora de seguir para um caminho mais flexível.
   - É necessário certo conhecimento para começar a usar o modelo porque a inexperiência com a forma de trabalhar do modelo pode levar a problemas posteriores.
3. *Modelo Incremental:* é uma estratégia de planejamento estagiado em que varias partes do sistema são desenvolvidas em paralelo e

integradas quando completas. As desvantagens do modelo Incremental são:

- Cada fase de uma iteração é rígida e não se sobrepõem umas às outras.
- Podem surgir problemas relativos à arquitetura do sistema, porque nem todos os requisitos estão reunidos na frente de todo o ciclo de vida do software.
- O modelo Incremental precisa ser relativamente pequeno.

4. *Modelo V:* é uma evolução do modelo em cascata. Uma de suas principais características é um fluxo linear e sequencial de atividades. As vantagens do modelo V são:

- A fase de testes é dividida entre as fases de análise e projeto.
- É eficiente em projetos de grande porte (por exemplo, projetos da área automotiva), pois sem a fase de teste as falhas são encontradas em etapas menos avançadas do projeto.

A desvantagem do modelo V é que:

- Se alguma mudança acontecer no meio do desenvolvimento, logo em seguida os documentos de testes, as analises e os documentos de requisitos deverão ser atualizados.

3.2 MODELO V DE DESENVOLVIMENTO DE SISTEMAS EMBARCADOS

O estudo se baseará no conceito do Modelo V para Desenvolvimento de Sistemas Embarcados, muito empregado na indústria automotiva. O modelo V permite que durante a integração de um sistema em seus diversos níveis, os testes sejam feitos contra os próprios requisitos do componente/interface que está sendo testado (a), ao contrario dos outros modelos (anteriores a este) onde o componente é testado contra a especificação do componente/interface. A figura 1 ilustra o modelo V simplificado.

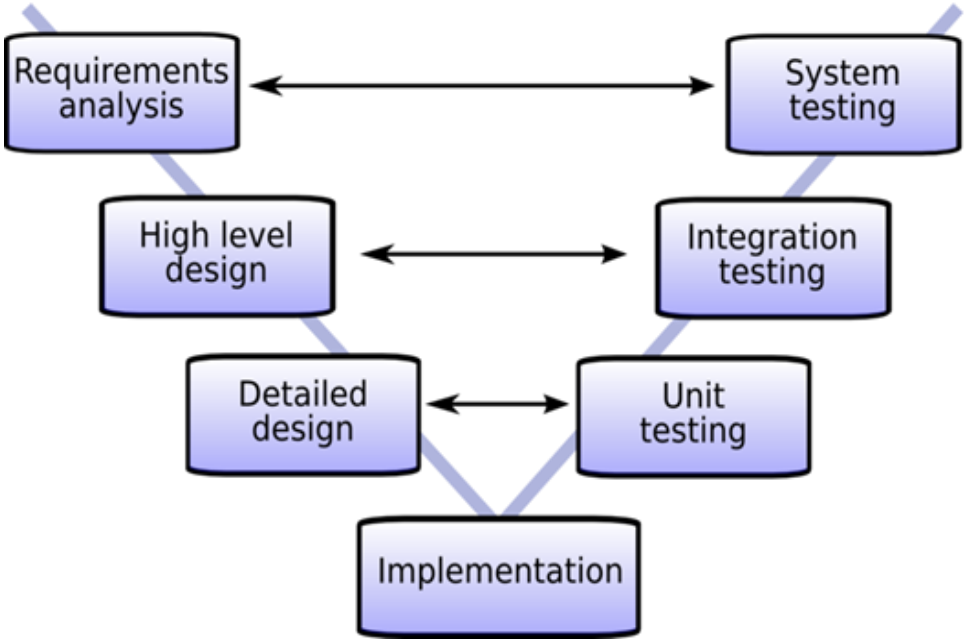

Figura 1 – Modelo V Simplificado de Desenvolvimento de Sistema Embarcado (WIKIWAND, 2016).

O Modelo V, que virou um padrão da indústria de software depois de 1980 e tornou-se um conceito padrão em todos os ramos da indústria, tem as seguintes características (PRESSMAN, 2002):

- Os testes têm resultados de maior efetividade;
- Possibilita que se encontrem erros durante os processos de se derivar especificações de requisitos;
- Ajuda a desenvolver novos requisitos;
- Melhora a qualidade do produto resultante.

Um desenvolvimento de produto de software, segundo o modelo V, tem as seguintes etapas (PRESSMAN, 2002):

1. Análise das necessidades e viabilidade;
2. Especificação do programa;
3. Concepção: arquitetura;
4. Concepção: detalhamento;
5. Codificação;
6. Teste individual;
7. Teste de integração;
8. Teste de validação;
9. Receita.

Assim, na figura 2 tem-se um modelo V mais detalhado e completo, contendo as etapas mencionadas acima no desenvolvimento de sistema embarcado.

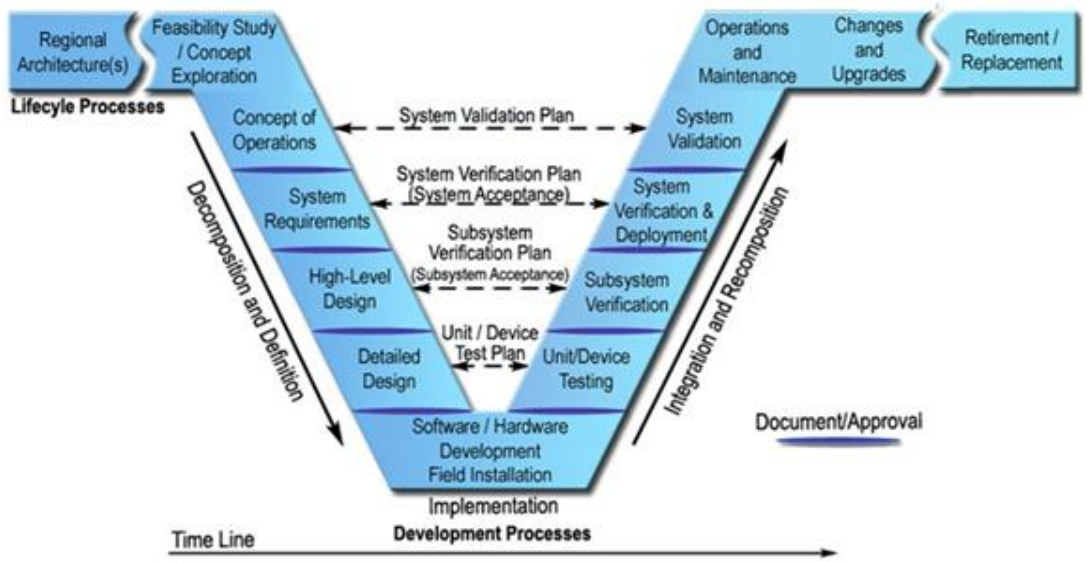

Figura 2 – Modelo V Detalhado de Desenvolvimento de Sistema Embarcado (FHWA, 2016).

3.3 VALIDAÇÃO DE SOFTWARE EMBARCADO

O teste de sistema pode ser definido como a aplicação de uma série de técnicas de teste e estratégias de desenvolvimento que visam o aumento da confiabilidade do programa. Tal objetivo é atingido quando diferentes aspectos do sistema em teste são examinados. O uso de ferramentas de análise estática permite a localização de potenciais problemas na forma como o programa foi construído e a execução de casos de teste bem construídos leva a uma verificação do atendimento dos requisitos de projeto. A análise de cobertura de teste permite a verificação do alcance dos testes executados, expondo regiões não testadas, ou devido a casos de teste mal projetados ou por requisitos de projeto mal documentados.

O processo de teste (também chamado de etapas do teste) envolve quatro atividades principais (PFLEEGER, 2004):

- *Planejamento de Testes:* trata da definição das atividades de teste, das estimativas dos recursos necessários para realizá-las, dos objetivos, estratégias e técnicas de teste a serem adotadas e dos

critérios para determinar quando uma atividade de teste está completa*;*

- *Projeto de Casos de Teste:* é a atividade chave para um teste bem-sucedido, ou seja, para se descobrir a maior quantidade de defeitos com o menor esforço possível. Os casos de teste devem ser cuidadosamente projetados e avaliados para tentar se obter um conjunto de casos de teste que seja representativo e envolva as várias possibilidades de exercício das funções do sistema (cobertura dos testes). Existe uma grande quantidade de técnicas de teste para apoiar os testadores a projetar casos de teste, oferecendo uma abordagem sistemática para o teste de SW*;*
- *Execução dos Testes ou do Programa com os Casos de Teste:* consiste na execução dos casos de teste e registro de seus resultados*;*
- *Analise ou Avaliação dos Resultados:* detectadas falhas, os defeitos deverão ser procurados. Não detectadas falhas, deve-se fazer uma avaliação final da qualidade dos casos de teste e definir pelo encerramento ou não de uma atividade de teste.

3.4 CASOS DE TESTE

Um Caso de Teste pode ser definido como um conjunto de entradas, condições de execução e um critério de sucesso ou falha (PEZZÉ; YOUNG, 2008). Em outras palavras, Caso de Teste é a especificação de uma entrada para o programa e a correspondente saída esperada. Nesta definição, entrada seria o conjunto de dados necessários para uma execução do SW, enquanto saída esperada é o resultado de uma execução do código. A busca por Casos de Teste interessantes é um dos mais importantes aspectos do teste de software. Caso de teste interessante é aquele que possui uma grande probabilidade de sucesso na localização de um eventual defeito ou falha (PFALLER, 2006). O projeto de casos de teste pode ser tão difícil quanto o projeto do próprio produto a ser testado.

3.4.1 Requisitos do Projeto

A primeira etapa de um procedimento de testes é a verificação do atendimento aos requisitos do projeto. A análise dos requisitos é a fonte mais utilizada para a geração de casos de teste. Uma abordagem de teste baseada nos requisitos é dita funcional, uma vez que se preocupa com a funcionalidade do programa. Entretanto, conforme Pfaller (2006), não existe ainda uma forma padronizada e definida de se derivar especificações de casos de teste a partir de requisitos de projeto especificados de maneira informal.

3.4.2 Verificação da Cobertura de Teste

A geração de casos de teste a partir da análise de cobertura provém da constatação de que não é possível aprovar a qualidade de partes do programa que não foram executadas. Seu objetivo primordial é a geração de casos de teste capazes de exercitar todas as estruturas internas do código.

3.4.3 Análise das Interfaces

Visando o teste de interfaces, existem os itens de teste, que são listas de aspectos a serem contemplados pelos testes, para interfaces de hardware e interfaces de sistema, chamado de EmITM (Embedded System’s Interface Test Model). Os itens de teste são agrupados conforme características testáveis, como memória, dispositivos de entrada e saída, timer, etc (SUNG; CHOI; SIN, 2007).

3.5 TÉCNICAS E CRITÉRIOS DE TESTE DE SOFTWARE

A atividade de teste de software é um elemento crítico da garantia de qualidade de SW e representa a última revisão de especificação, projeto e codificação.

3.5.1 Objetivos da Atividade de Teste

1. A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro;
2. Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;

3. Um teste bem sucedido é aquele que revela um erro ainda não descoberto.

3.5.2 Tipos de Teste de Software

Para testar um módulo, é necessário definir um caso de teste, executar o módulo com os dados de entrada definidos por esse caso de teste e analisar a saída. Um teste é um conjunto limitado de casos de teste, definido a partir do objetivo do teste (PFLEEGER, 2004). Diversas técnicas de teste têm sido propostas visando apoiar o projeto de casos de teste. Essas técnicas podem ser classificadas, segundo a origem das informações utilizadas para estabelecer os objetivos de teste, em, dentre outras categorias, técnicas funcional, estrutural ou baseadas em máquinas de estado (MALDONADO, 2001). Os testes funcionais ou Caixa Preta utilizam as especificações (de requisitos, análise e projeto) para definir os objetivos do teste e, portanto, para guiar o projeto de casos de teste. O conhecimento sobre uma determinada implementação não é usado (MALDONADO, 2001). Assim, os testes são conduzidos na interface do programa. Os testes Caixa Preta são empregados para demonstrar que as funções do sistema estão operacionais, que a entrada é adequadamente aceita e a saída é corretamente produzida e que a integridade da informação externa (uma base de dados, por exemplo) é mantida (PRESSMAN, 2002). Os testes estruturais ou Caixa Branca estabelecem os objetivos do teste com base em uma determinada implementação, verificando detalhes do código. Caminhos lógicos internos são testados, definindo casos de testes que exercitem conjuntos específicos de condições ou laços (PRESSMAN, 2002). Os testes baseados em máquinas de estado são projetados utilizando o conhecimento subjacente à estrutura de uma máquina de estados para determinar os objetivos do teste (MALDONADO, 2001). É importante ressaltar que técnicas de teste devem ser utilizadas de forma complementar, já que elas têm propósitos distintos e detectam categorias de erros distintas (MALDONADO, 2001). De inicio, pode parecer que realizando testes caixa branca rigorosos poderíamos chegar a programas corretos. Contudo, isso não é prático, uma vez que todas as combinações possíveis de caminhos e valores de variáveis teriam de ser exercitadas, o que é impossível. Isso não quer dizer, entretanto, que os testes Caixa Branca não são úteis. Testes Caixa Branca podem ser usados, por exemplo, para

garantir que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez (PRESSMAN, 2002).

3.5.3 Teste Caixa Branca

Existem três técnicas de teste Caixa Branca, cada uma delas procurando apoiar o projeto de casos de teste focando em algum ou vários aspectos da implementação (PRESSMAN, 2002):

- *Técnica Funcional:* Para testar os requisitos funcionais do sistema. A técnica utiliza os critérios de Particionamento em Classes de Equivalência, Analise do Valor Limite e Grafo de Causa-Efeito*;*
- *Técnica Estrutural:* Para testar a estrutura interna do código. A técnica usa critérios Estruturais baseados em Fluxo de Dados*;*
- *Técnica Baseada em Erros:* Para testar os erros mais frequentes cometidos durante o processo de desenvolvimento de programa. A técnica utiliza critérios da Análise de Mutantes, também chamados de critérios Baseados em Mutação.

3.5.3.1 Técnica funcional

Esta técnica é usada para testar os requisitos funcionais do sistema e ela utiliza os seguintes critérios:

- *Particionamento de Equivalência:* Este método divide o domínio de entrada de um módulo em classes de dados ou de equivalência, a partir das quais os casos de teste podem ser derivados. O particionamento de equivalência procura definir um caso de teste que descubra classes de erros, reduzindo assim o número total de casos de teste que devem ser desenvolvidos e ficando apenas com um caso de teste para cada classe, uma vez que, todos os elementos de uma mesma classe devem se comportar de maneira equivalente. Uma classe de equivalência representa um conjunto de estados válidos para condições de entrada (PRESSMAN, 2002).

- *Análise de Valor Limite BVA:* A prática mostra que um grande número de erros tende a ocorrer nas fronteiras do domínio de entrada de um módulo. Tendo isso em mente, a análise de valor limite leva à escolha de casos de teste que põem à prova os valores fronteiriços. A análise de valor limite é uma técnica de projeto de casos de teste que complementa o particionamento de equivalência. Em vez de selecionar qualquer elemento de uma classe de equivalência, a BVA leva à seleção de casos de testes nas extremidades da classe. Em vez de se concentrar somente nas condições de entrada, a BVA deriva os casos de teste também do domínio de saída (PRESSMAN, 2002).

3.5.3.2 Testes de estrutura de controle ou técnica estrutural

Como o próprio nome diz, enfocam as estruturas de controle de um módulo, tais como comandos, condições e laços. Aplicada a Testes Caixa Branca, é uma técnica baseada no conhecimento da estrutura interna (implementação) do programa. Com esta técnica é possível testar detalhes dos procedimentos (PRESSMAN, 2002).

Estes testes ampliam a cobertura dos testes e melhoram a qualidade dos testes (MALDONADO, 2001):

- *Teste de Condição:* é um tipo de teste de estrutura de controle que avalia as condições lógicas contidas num módulo de programa. O teste de condição concentra-se em testar as condições do código.
- *Teste de Fluxo de Dados:* testa caminhos de teste (fluxo de dados) de um código de acordo com as localizações das definições e dos usos das variáveis nos módulos.
- *Teste de Ciclos ou Laços:* avalia exclusivamente as construções de laços (loops) simples, aninhados, concatenados e não estruturados.

A maioria dos critérios desta técnica utiliza uma representação de código conhecida como grafo de programa ou grafo de fluxo de controle. A figura 3 ilustra um exemplo de Grafo de Programa.

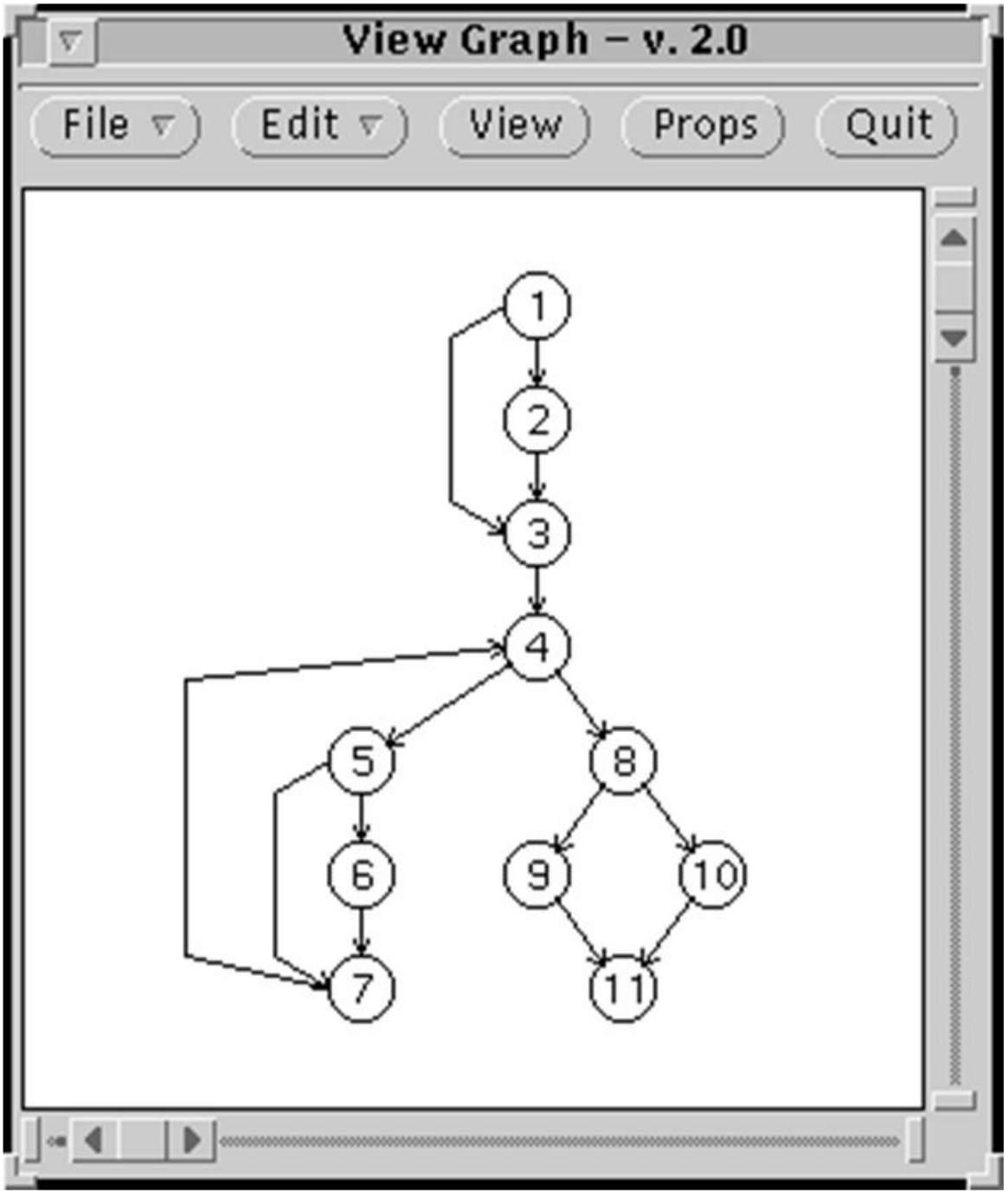

Figura 3 – Grafo de Programa do identifier gerado pela View-Graph (BARBOSA, 2000).

3.5.3.3 Técnica baseada em erros

Os requisitos de teste são derivados a partir dos erros mais frequentes cometidos durante o processo de desenvolvimento do programa (MALDONADO, 2001). Os critérios desta técnica são Semeadura de Erros e Teste de Mutação. O Teste de Mutação pode ser uma Analise de Mutantes (utilizada em Testes de Unidade) ou um teste de Mutação de Interface (utilizada em Testes de Integração).

3.5.3.4 Outras técnicas de teste caixa branca

Segundo Maldonado (2001), existem ainda outras técnicas de teste caixa branca, como, por exemplo, a técnica do Teste de Caminho Básico:

- *Teste de Caminho Básico**:*** O método possibilita que o projetista do caso de teste derive ou defina uma medida da complexidade lógica de um modulo e use essa medida como guia para definir um conjunto básico de caminhos de execução (MALDONADO, 2001).

3.5.4 Teste Caixa Preta

Assim como há diversas técnicas de teste caixa branca, o mesmo acontece em relação ao teste caixa preta. Dentre as diversas técnicas de teste caixa preta, podem ser citadas:

- *Técnica Funcional.*
- *Teste de Sistemas de Tempo Real.*

3.5.4.1 Técnica funcional

Esta técnica é aplicada a Testes Caixa Preta e ela baseia-se na especificação do software para derivar os requisitos de teste e aborda o SW de um ponto de vista macroscópico, ou seja, com uma visão de sistema. Por esta razão os Testes Caixa Preta (da Técnica Funcional) também são chamados de Testes de Sistema (MALDONADO, 2001).

A Técnica Funcional envolve duas etapas principais:

1. Identificar as funções que o programa deve realizar (especificação dos requisitos);
2. Criar casos de teste capazes de checar se essas funções estão sendo executadas corretamente;

Os problemas da Técnica Funcional são dificuldades em quantificar a atividade de teste, uma vez que não se pode garantir que partes essenciais ou criticas do código foram executadas, e de automatização.

3.5.4.2 Teste de sistemas de tempo real

Neste tipo de teste, encontramos o elemento de combinação alinhado ao teste que é o tempo. Os testes de software devem levar em consideração o impacto das falhas de hardware sobre o processamento do SW. Tais falhas podem ser extremamente difíceis de ser simuladas realisticamente (PRESSMAN, 2002).

Estratégia global para sistemas de tempo real (PRESSMAN, 2002):

1. Teste de tarefas: o primeiro passo da atividade de testes de tempo real consiste em testar cada tarefa independentemente. Este teste revela erros de lógica e de função, mas não revelará erros comportamentais ou de timing;
2. Teste comportamental: é possível simular o comportamento de um sistema de tempo real e examinar seu comportamento como uma consequência de eventos externos. O comportamento do programa é examinado a fim de detectar erros de comportamento;
3. Teste intertarefas: as tarefas assíncronas, que se comunicam entre si, são testadas com diferentes taxas de dados e cartas de processamento para determinar se ocorrerão erros de sincronização intertarefas;
4. Teste do sistema: o SW e o HW são integrados e uma variedade completa de testes de sistema é realizada, numa tentativa de descobrir erros na interface programa/ hardware.

3.6 ESTRATÉGIAS DE TESTE DE SOFTWARE

Um esqueleto (template) de teste de software deve ser definido para o processo de engenharia do SW. Esqueleto é um conjunto de passos no qual podemos alocar técnicas de projeto de casos de teste e métodos de teste específico.

O projeto efetivo de casos de teste é importante, mas não suficiente para o sucesso da atividade de testes. A estratégia, isto é, a série planejada de realização dos testes, é também crucial (PRESSMAN, 2002).

Uma estratégia de teste de sistema deve acomodar testes de baixo nível (Testes Caixa Branca) que seja necessário para verificar se um segmento de

código-fonte foi corretamente implementado, bem como testes de alto nível (Testes Caixa Preta) que valide funções importantes do sistema contra requisitos do cliente.

3.6.1 Verificação, Validação e Teste de Software

A fim de garantir a qualidade de um programa, são usados os processos de Verificação, Validação e Teste. A verificação refere-se ao conjunto de atividades que garante que o SW implemente corretamente uma função específica. A validação refere-se a um conjunto diferente de atividades que garante que o programa que foi construído é rastreável às exigências do cliente. Já o processo de Testes é para examinar o comportamento do produto por meio de sua execução.

Utilizando estes processos pode-se encontrar no sistema defeitos, erros ou falhas. Em software, defeito é uma deficiência mecânica ou algorítmica que, se ativada, pode levar a uma falha. Já Erro é definido como um item de informação ou estado de execução inconsistente. Por outro lado, Falha é um evento notável em que o sistema viola suas especificações.

Os métodos de análise, projeto e implementação (codificação) atuam no sentido de aumentar a qualidade ao oferecer técnicas uniformes e resultados previsíveis. A análise e o projeto de software (juntamente com a codificação) são tarefas construtivas, ou seja, o engenheiro de SW cria um programa de computador, sua documentação e estruturas de dados correlatas. Do ponto de vista do construtor, a atividade de teste pode ser destrutiva, pois sempre vão ser descobertos erros no código.

Em algumas situações, no mundo dos programas, Teste é confundido com Depuração. Mas enquanto Teste é um processo de execução de um programa com o objetivo de revelar a presença de erros e ainda que contribui para aumentar a confiança de que o SW desempenha as funções especificadas, Depuração é a consequência não previsível do teste. Ainda na Depuração, depois de revelada a presença do erro, este deve ser encontrado e corrigido.

Na maioria das estratégias de teste de software existem três grandes fases de Teste bem definidas (MALDONADO, 2001):

1. *Teste de Unidade*: Também chamado de Teste Unitário, de Módulo ou Caixa Branca, tem por objetivo testar a menor unidade do projeto (um componente de programa que não pode ser subdividido),

procurando identificar erros de lógica e de implementação em cada módulo do sistema, separadamente. Concentra-se em cada unidade de SW de acordo com a implementação do código-fonte e exercita caminhos específicos da estrutura de controle de um módulo, a fim de garantir uma completa cobertura e máxima detecção de erros;

2. *Teste de Integração*: A atenção encontra-se no projeto e na construção da arquitetura do código. Este teste cuida das questões associadas aos duplos problemas da verificação e construção dos programas. Nesta fase tenta-se identificar erros associados às interfaces entre os módulos quando esses são integrados para formar a estrutura do sistema do produto;
3. *Teste de Sistema*: Também chamado de Teste Funcional ou Caixa Preta. Neste teste o código e outros elementos do sistema são testados como um todo. É verificado se todos os elementos do sistema combinam-se adequadamente e se a função ou desempenho global do sistema é conseguida. Nesta fase é onde se verifica se as funções estão de acordo com a especificação;

Mas existe uma estratégia de teste de software na qual uma quarta fase de teste é inserida no contexto. Esta fase seria a de *Teste de Validação.*

3.6.2 Teste de Unidade

O teste unitário concentra-se no esforço de verificação da menor unidade de projeto de programa - o módulo. A menor unidade refere-se a um procedimento ou função. Caminhos de controle importantes são testados para descobrirem erros dentro das fronteiras do módulo.

A interface com o módulo é testada para ter a garantia de que as informações fluem para dentro e para fora da unidade de programa que se encontra sob teste. A estrutura de dados local é examinada para ter a garantia de que os dados armazenados temporariamente mantêm sua integridade durante todos os passos de execução de um algoritmo. As condições de limite são testadas para ter a garantia de que o módulo opera adequadamente nos limites estabelecidos para demarcarem ou restringirem o processamento.

Entre os erros mais comuns estão:

- Precedência aritmética incorreta ou mal compreendida;
- Operações em modo misto;
- Inicialização incorreta.

Os casos de teste devem descobrir erros tais como:

- Comparação de diferentes tipos de dados;
- Operadores lógicos ou precedência incorretos;
- Término do laço impróprio ou inexistente.

3.6.3 Teste de Integração

Este teste é uma técnica sistemática para a construção da estrutura de código realizando-se, ao mesmo tempo, testes para descobrir erros associados a interfaces. O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.

Frequentemente, existe uma tendência para tentar a integração não incremental; ou seja, construir o código usando uma abordagem "big bang" onde todos os módulos são combinados antecipadamente. O SW completo é testado como um todo.

A integração incremental é a antítese do big bang. O programa é construído e testado em pequenos segmentos, onde os erros são mais fáceis de ser isolados e corrigidos, as interfaces têm maior probabilidade de ser testadas completamente e uma abordagem sistemática ao teste pode ser aplicada. Um caminho independente é qualquer caminho ao longo de um módulo que introduz pelo menos um novo comando de processamento ou condição (PRESSMAN, 2002).

Integração Top-Down: é uma abordagem incremental à construção das estruturas de código. Os módulos são integrados movimentando-se de cima para baixo através da hierarquia de controle, iniciando-se do módulo de controle principal (programa principal). Os módulos subordinados ao módulo de controle principal são incorporados à estrutura de uma maneira depht-first (primeiramente pela profundidade) ou breadth-first (primeiramente pela largura).

Integração Bottom-Up: inicia a construção e os testes com módulos atômicos (isto é, módulos localizados nos níveis mais baixos da estrutura de código). Uma vez que os módulos são integrados de baixo para cima, o processamento

exigido para os módulos subordinados em determinado nível está sempre disponível, e a necessidade de stubs (dublês de teste) é eliminada.

A maior desvantagem da abordagem top-down é a necessidade de ter stubs e as dificuldades de teste resultantes que podem ser compensados pela vantagem de testar logo as principais funções de controle. A maior desvantagem da integração bottom-up é que o programa não existe como entidade até que o último módulo seja adicionado.

Os critérios a seguir são aplicados a todas as fases de testes:

Integridade de interface: As interfaces internas e externas são testadas à medida que cada módulo é incorporado à estrutura.

Validade funcional: Testes projetados para a descoberta de erros funcionais.

Conteúdo de informação: Testes projetados para a descoberta de erros associados às estruturas de dados globais e locais.

Desempenho: Testes projetados para a verificação dos limites de desempenho estabelecidos durante o projeto do programa.

3.6.4 Teste de Sistema

O teste de sistema é na verdade, uma série de diferentes testes, cujo propósito primordial é por completamente à prova o sistema baseado em computador. Não obstante cada teste tenha uma finalidade diferente, todo o trabalho deve verificar se todos os elementos do sistema foram adequadamente integrados e realizam as funções atribuídas. Este teste tem por objetivo identificar erros de funções (requisitos funcionais) e características de desempenho (requisito não funcional) que não estejam de acordo com as especificações.

Os testes de sistema incluem diversos tipos de teste, realizados na seguinte ordem (PFLEEGER, 2004):

1. *Teste Funcional:* verifica se o sistema integrado realiza as funções especificadas nos requisitos*;*
2. *Teste de Desempenho:* É idealizado para testar o desempenho de run-time do programa dentro do contexto de um sistema integrado e também verifica se o sistema integrado atende os requisitos não funcionais (eficiência, segurança, confiabilidade etc). O teste de

desempenho ocorre ao longo de todos os passos do processo de teste*;*

3. *Teste de Aceitação:* os testes funcionais e de desempenho são ainda realizados por desenvolvedores, entretanto é necessário que o sistema seja testado pelos clientes. No teste de aceitação, os clientes testam o sistema a fim de garantir que o mesmo satisfaz suas necessidades. Vale destacar que o que foi especificado pelos desenvolvedores pode ser diferente do que queria o cliente. Assim, o teste de aceitação assegura que o que foi solicitado é o que foi construído*;*
4. *Teste de Instalação:* algumas vezes o teste de aceitação é feito no ambiente real de funcionamento, outras não. Quando o teste de aceitação for feito em um ambiente de teste diferente do local em que será instalado, é necessário realizar testes de instalação.

Outros tipos de testes de sistema:

Teste de Recuperação: o teste de recuperação é um teste que força o programa falhar de diversas maneiras e verifica se a recuperação é adequadamente executada.

Teste de Segurança: este teste tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão, de acessos indevidos. Durante o teste de segurança, o analista desempenha papéis de pessoas que desejam penetrar no sistema, qualquer coisa vale, tentando desarma-lo, e derrubar as defesas que tenham sido construídas.

3.6.5 Teste de Validação

No Teste de Validação, também conhecido como Teste de Desempenho ou ainda Teste de Aceitação, é onde os requisitos estabelecidos como parte da análise de requisitos de programa são validados em relação ao SW que foi construído. O teste de validação oferece a garantia final de que o sistema atende a todas as exigências funcionais, comportamentais e de desempenho.

Tomando por base essas fases, a atividade de teste pode ser estruturada de modo que, em cada fase, diferentes tipos de erros e aspectos do programa sejam considerados (MALDONADO, 2001). Tipicamente, os primeiros

testes focalizam componentes individuais e aplicam testes Caixa Branca e Caixa Preta para descobrir erros. Após os componentes individuais terem sido testados, eles precisam ser integrados, até se obter o sistema por inteiro. Na integração, o foco é o projeto e a arquitetura. Finalmente, uma série de testes de alto nível é executada quando o sistema estiver operacional, visando a descobrir erros nos requisitos (PFLEEGER, 2004). No teste de unidade, faz-se necessário construir pequenos componentes para permitir testar os módulos individualmente, os ditos drivers e stubs. Um driver é um programa responsável pela ativação e coordenação do teste de uma unidade e ele é responsável por receber os dados de teste fornecidos pelo testador, passar esses dados para a unidade sendo testada, obter os resultados produzidos por essa unidade e apresentá-los ao testador. Um stub é uma unidade que substitui, na hora do teste, uma outra unidade chamada pela unidade que está sendo testada. Em geral, um stub simula o comportamento da unidade chamada com o mínimo de computação ou manipulação de dados (MALDONADO, 2001).

A abordagem de integração de módulos pode ter impacto na quantidade de drivers e stubs a ser construída.

3.7 CENÁRIOS DE TESTE

A execução prática de um caso de teste requer a criação deste dentro de um programa chamado driver de teste. Este SW é o responsável por criar todo o contexto necessário para a execução do caso de teste em um ambiente que simula as condições reais de execução do módulo. O driver também é o responsável pela coleta e análise dos resultados de teste, gerando relatórios quando necessário.

3.8 DUBLÊS DE TESTE (MODELOS)

Quando hardwares ou componentes reais podem não estar disponíveis no momento do teste mecanismos de abstração de tais recursos são fundamentais. Segundo Meszaros (2009), modelos ou dublês de teste são componentes que não

se comportam exatamente como os componentes reais aos quais eles substituem, mas provém a mesma interface, de forma que o programa em teste não o diferencie do componente real.

3.9 FERRAMENTAS DE DESENVOLVIMENTO E TESTE DE SOFTWARE

A automatização, o quanto for possível, do processo de desenvolvimento e teste de SW é uma necessidade por parte das empresas, uma vez que, tanto os tempos de desenvolvimento quanto as margens de lucro estão cada vez menores. Isto faz com que a busca por ferramentas de desenvolvimento e teste seja constante. O uso adequado de tais ferramentas pode multiplicar a produtividade de uma equipe de desenvolvimento.

3.9.1 Visualizadores de Código e Métricas de Qualidade de Software

Métricas de qualidade de código são obtidas a partir da análise da estrutura interna do programa. Uma correta organização da estrutura interna do SW permitirá uma grande redução da probabilidade de existência de defeitos. O uso de visualizadores eficientes de código, além de facilitar o desenvolvimento do SW, permite uma melhor compreensão do que está sendo construído (SCITOOLS, 2016).

3.9.2 Análise Estática de Código

Técnicas de análise estática de código obtêm informação a respeito do comportamento do programa baseado na sua representação estática, diferentemente da análise dinâmica, que precisa executar o código para a sua verificação e que, por isto, requer o uso de casos de teste. Analisadores estáticos podem rastrear inconsistências e redundâncias entre módulos do SW (PC-LINT, 2016).

3.9.3 Análise de Cobertura de Teste

As ferramentas de análise de cobertura de teste têm por objetivo a verificação, de forma automática, do alcance dos casos de teste executados. A execução dos casos de teste funcionais não é suficiente para a verificação completa do sistema. Somente a análise de cobertura do teste será capaz de rastrear se todos

os nós, laços ou condições lógicas do código foram contemplados por ao menos um caso de teste (MALDONADO, 2001).

3.9.4 Simuladores

Apesar dos ambientes de desenvolvimento de sistemas embarcados fornecerem simuladores para o conjunto de instruções do microcontrolador, tais simuladores não englobam os periféricos de hardware. Ou seja, sempre que uma instrução ou função do módulo depender de uma resposta de um periférico de hardware para seguir em frente, ela irá travar ou responder de forma inesperada (SEO et al., 2007).

3.9.5 Emuladores, Placas de Avaliação e Protótipos

Devido à forte interdependência entre SW embarcado e HW, emuladores, placas de avaliação e protótipos são muito utilizados com o objetivo principal de lidar, desde as primeiras etapas de projeto, com um ambiente de desenvolvimento e teste que possua um comportamento o mais próximo o possível do que será verificado no sistema embarcado definitivo (SEO et al., 2007).

3.10 ESTUDO TEÓRICO E APLICADO DE CRITÉRIOS DE TESTE E VALIDAÇÃO NA PRODUÇÃO DE SOFTWARE

Este trabalho consiste em estudo teórico e empírico, definição e comparação de critérios de teste e implementação de ferramentas de software de apoio às atividades de teste na produção de software. Atividades de comparação entre critérios de teste funcional e estrutural, com ênfase em critérios baseados em análise de fluxo de dados e em critérios baseados em erros, com ênfase na Análise de Mutantes, também são conduzidos dentro do escopo desta linha. Todos esses aspectos são explorados tanto no nível de teste de unidade como no teste de integração, assim como à luz do paradigma de desenvolvimento de software orientado a objeto (MALDONADO, 2001).

3.11 AUTOMATIZAÇÃO DA ATIVIDADE DE TESTE

Para a aplicação efetiva de um critério de teste é necessário o uso de ferramentas automatizadas de teste que apoiem a aplicação desse critério. Automatizações de teste ainda facilitam a condução de estudos comparativos entre critérios de teste. Para automatização de atividades de teste são utilizados ferramentas de teste como os xSuds (em inglês, Software Understanding & Diagnosis System). São exemplos de xSuds (SEO et al., 2007):

- xAtac: para automatização de testes;
- xSlice: para automatização de processos de depuração;
- xDiff: para automatização de comparações de códigos.

4 ENTREGA E MANUTENÇÃO DE SOFTWARE

Concluídos os testes, sistema aceito e instalado, o processo de desenvolvimento de software está chegando ao fim. A entrega é a última etapa deste processo. Uma vez entregue, o sistema passa a estar em operação e eventuais alterações, de caráter corretivo ou de caráter de evolução, caracterizam-se como uma manutenção.

4.1 ENTREGA

A entrega não é meramente uma formalidade. No momento em que o sistema é instalado no local de operação e devidamente aceito, é necessário, ainda, ajudar os usuários a entenderem e a se sentirem mais familiarizados com o sistema. Neste momento, duas questões são cruciais para uma transferência bem-sucedida: treinamento e documentação (PFLEEGER, 2004).

A operação do sistema é extremamente dependente de pessoal com conhecimento e qualificação. Portanto, é essencial que o treinamento de pessoal seja realizado para que os usuários e operadores possam operar o sistema adequadamente.

A documentação que acompanha o sistema também tem papel crucial na entrega, afinal ela será utilizada como material de referência para a solução de problemas ou como informações adicionais. Essa documentação inclui, dentre outros, manuais do usuário e do operador, guia geral do sistema, tutoriais, ajuda (help), preferencialmente on-line e guias de referência rápida (PFLEEGER, 2004).

4.2 MANUTENÇÃO

O desenvolvimento de um sistema termina quando o produto é entregue para o cliente e entra em operação. A partir daí, deve-se garantir que o sistema continuará a ser útil e atendendo às necessidades do usuário, o que pode demandar alterações no mesmo. Começa, então, a fase de manutenção.

Há muitas causas para a manutenção, dentre elas: falhas no processamento devido a erros no software, falhas de desempenho, alterações no ambiente de dados, alterações no ambiente de processamento, necessidade de modificações em funções existentes e necessidade de inclusão de novas capacidades.

Ao contrário do que podemos pensar, a manutenção não é uma atividade trivial nem de pouca relevância. Ela é uma atividade importantíssima e de intensa necessidade de conhecimento. O mantenedor precisa conhecer o sistema, o domínio de aplicação, os requisitos do sistema, a organização que utiliza o mesmo, práticas de engenharia de software passadas e atuais, a arquitetura do sistema, algoritmos usados etc.

O processo de manutenção é semelhante, mas não igual, ao processo de desenvolvimento e pode envolver atividades de levantamento de requisitos, análise, projeto, implementação e testes, agora no contexto de um software existente. Essa semelhança pode ser maior ou menor, dependendo do tipo de manutenção a ser realizada.

Segundo Pfleeger (2004), existem os seguintes tipos de manutenção:

- *Manutenção corretiva*: trata de problemas decorrentes de defeitos. À medida que falhas ocorrem, elas são relatadas à equipe de manutenção, que se encarrega de encontrar o defeito que causou a falha e faz as correções (nos requisitos, análise, projeto ou implementação), conforme o necessário. Esse reparo inicial pode ser temporário, visando manter o sistema funcionando. Quando esse for o caso, mudanças mais complexas podem ser implementadas posteriormente.
- *Manutenção adaptativa*: às vezes, uma mudança no ambiente do sistema, incluindo hardware e software de apoio, pode implicar em uma necessidade de adaptação.
- Manutenção perfectiva: consiste em realizar mudanças para melhorar algum aspecto do sistema, mesmo quando nenhuma das mudanças for consequência de defeitos. Isso inclui a adição de novas capacidades bem como ampliações gerais.

- Manutenção preventiva: consiste em realizar mudanças a fim de prevenir falhas. Geralmente ocorre quando um mantenedor descobre um defeito que ainda não causou falha e decide corrigi-lo antes que ele gere uma falha (PFLEEGER, 2004).

5 ESTUDO DE CASO

Testes de sistema durante o desenvolvimento de produtos são práticas comuns e necessárias, que por vezes são realizadas de forma superficial ou até mesmo erroneamente ignoradas, devido aos períodos de desenvolvimento de produtos cada vez mais reduzidos, à grande competição entre as montadoras de veículos e à consequente necessidade de ser a primeira a lançar no mercado um novo produto.

Este trabalho apresenta um caráter exploratório, onde se buscou investigar a aplicação de métodos, processos e aplicação de um modelo de referência na validação de software no desenvolvimento de novos produtos de sistema embarcado.

5.1 CONSIDERAÇÕES INICIAIS

Um modelo de referência de processo serve para orientar a implantação de um processo em uma organização, com o intuito de torna-lo repetível e possível de ser medido. Neste contexto, o modelo escolhido e apresentado ao longo do trabalho tem o objetivo de orientar a implantação de um processo de desenvolvimento de software.

Como já mencionado no capítulo 3, o modelo de referência é muito utilizado em diversas empresas do segmento automotivo, sobretudo as sistemistas (de Autopeças) e as montadoras de veículos (carro, veículos comerciais, ônibus e caminhões) e de maquinas industriais (maquinas agrícolas e de construção), que desenvolvem tecnologia de ponta e tem em seus portfólios produtos com software embarcado. Porém neste capítulo é descrito um estudo de caso onde o modelo de referência foi utilizado com sucesso, após ser refinado e melhorado, em uma montadora do setor agrícola. O estudo de caso teve o intuito de utilizar o modelo de referência para mostrar a evolução na utilização do modelo e as aplicações com sucesso de algumas metodologias e processos de validação de software embarcado empregados nesta empresa.

5.2 METODOLOGIA DE VALIDAÇÃO DE SOFTWARE APLICADA NO SETOR AUTOMOTIVO

Ao longo do período de desenvolvimento de sistema embarcado existem vários ciclos de validação, geralmente pré-definidos e programados pelo Gerente de Testes, com meses de antecedência, nas reuniões de definição de Plano de Validação do Projeto. O objetivo principal nestes ciclos de validação é identificar o mais cedo possível eventuais anomalias ou bugs, muito comuns principalmente nos primeiros meses de implementação do código.

Há algumas décadas a montadora do setor agrícola em questão tinha em seu portfólio produtos essencialmente mecânicos. Não havia softwares para controlar ou gerenciar funções ou sistemas. Havia apenas alguns sistemas elétricos como sistemas de iluminação e sinalização nos veículos. Logo, não havia modelo, processos definidos ou metodologias de validação de SW. Com a mecanização do setor agrícola, a exigência dos clientes por produtos melhores e mais eficientes no campo, a evolução da eletrônica no setor automotivo, a inserção de novas tecnologias e aumento da concorrência no setor agrícola, fizeram com que os produtos da empresa passassem a ter mais sistemas eletroeletrônicos, que com o tempo evoluíram para sistemas eletrônicos com código embarcado, que passaram a demandar a necessidade de testes eletrônicos e de software. Os profissionais da área de Validação de Produto da empresa, responsáveis por testar estes novos softwares, precisariam se adaptar a uma nova forma de testes, já que estavam acostumados a realizar apenas testes de funções mecânicas ou hidráulicas. Daí a necessidade de se contratar também mais pessoas da área de Eletrônica. Como no inicio dos testes de sistema tudo era feito sem procedimento (não havia ainda um processo e/ou uma metodologia de validação de SW durante o desenvolvimento dos primeiros projetos que continham sistemas embarcados), começou a surgir a necessidade de se ter um processo, uma metodologia e um modelo para avaliar os códigos embarcados. Foi aí que o modelo V de desenvolvimento de software e boa parte dos processos e metodologias de validação de código embarcado (associados ao modelo V) passaram a ser aplicados sistematicamente também em testes, na validação de produto. No inicio houve certa dificuldade de adaptação com os processos e metodologias e falhas comuns no inicio de qualquer introdução de uma

nova sistemática. Mas tanto os processos quanto as metodologias de testes foram sendo aprimoradas, atualizadas e complementadas com novas técnicas, conforme descrito logo adiante, que contribuíram para a confiabilidade, excelência, qualidade e maturidade dos testes de software realizados pelo time de validação do produto.

Nesta empresa, após o modelo V de desenvolvimento de software atingir um elevado nível de eficiência, excelência, aplicabilidade e confiabilidade, passou a existir, ao longo de todo o ano e paralelamente a outros sites do grupo espalhados ao redor do mundo, três grandes períodos de Validação de Sistema Embarcado pré-definidos. Cada período de Validação dura pelo menos dois meses e tem pelo menos dois ciclos de testes, com duração de dias ou semanas, dependendo da severidade e/ou da quantidade de bugs encontrados no início e ao longo do período de testes.

Os desenvolvedores do grupo, incluindo os terceirizados, além de desenvolverem os códigos são responsáveis pelos Testes de Unidade (ou de Módulo) e de Integração, ambos ainda no ambiente de desenvolvimento, ou seja, interagindo com o código (Testes Caixa Branca). Se não for encontrados bugs nestes testes o pacote de SW é liberado para o time de Teste de Sistema ou Funcionais da Engenharia de Desenvolvimento, que se encarregará de realizar a próxima etapa dos testes, os Testes Funcionais (Testes Caixa Preta) em bancada, ou seja, em um ambiente simulado e/ou emulado, tanto por SW quanto por HW. Se não for identificado nenhum bug crítico (eventuais bugs de baixa severidade poderão ser tolerados nestes testes e tratados mais adiante, para serem verificados e testados novamente nos próximos períodos de validação) o pacote de programas e Firmware será liberado e disponibilizado para os Engenheiros de Testes do time de Validação de Produto, para estes realizarem os Testes de Aceitação e mais Testes Funcionais, todos agora em veículo, em Laboratório e/ou em Campo, dando início ao período de Validação de Sistema Embarcado mencionado no parágrafo anterior. Neste período de validação serão realizados Testes Funcionais e Testes de Aceitação, também sem nenhuma interação com o código, sendo, portanto, também Testes Caixa Preta. Se for identificado algum bug de média ou alta severidade durante o período de Testes Funcionais no Veículo este problema será reportado aos Engenheiros de Software da Engenharia de Desenvolvimento, que tentarão corrigir os bugs identificados o

mais rápido possível, para que todo o processo descrito anteriormente seja reiniciado novamente (começando pelos Testes de Unidade), dando início a um novo ciclo de testes, dentro ainda do atual período de Validação ou de Testes de Aceitação. Problemas de SW de baixíssima ou baixa severidade, encontrados durante o período de Testes Funcionais no Veículo, serão tratados para os próximos períodos de validação. Um produto de Sistema Embarcado, ou uma versão de software (ou ainda um pacote de SW) será liberado para a produção quando nenhum bug for encontrado (ou quando apenas bugs de baixíssima severidade forem encontrados) durante o período de Testes nos Veículos.

Após a introdução de técnicas e boas práticas mencionadas ao longo deste artigo, e já muito difundidas em outras empresas com mais anos de experiência em desenvolvimento de sistemas embarcados, as validações mais recentes melhoraram consideravelmente em termos de redução de tempo na realização dos testes, e uma consequência disso foram reduções de custos durantes os testes, principalmente de viagens para o campo (devido à redução do numero de viagens e de menos dias de permanência em campo), em comparação com os primeiros testes, que eram realizados praticamente sem nenhuma metodologia e por pessoas que não eram da área de Eletrônica ou de Software. Outra melhora significativa foi na qualidade dos testes, tanta na execução dos testes em si quanto na apresentação e divulgação dos dados, que se tornaram mais confiáveis e precisos. Mais bugs, difíceis de serem identificados, foram encontrados e num intervalo de tempo menor do que era quando não se tinha um processo estabelecido.

Não se tem dados precisos que comprovam esta evolução, mas nas ultimas décadas as áreas de atuação da empresa no setor agrícola aumentaram bastante. A empresa, que anteriormente tinha em sua linha de produtos, alguns modelos de tratores e de colheitadeiras de grãos bem conceituados no mercado mas essencialmente mecânicos, passou a ter em seu portfólio produtos como pulverizadores auto propelidos, colheitadeiras de cana de açúcar e de café e plantadeiras (controladas através de software), ou seja, produtos com alta evolução tecnológica, com sistemas funcionais bastante complexos e com muito software embarcado. E não resta duvida que a introdução de metodologias e técnicas de

validação de software e a constante evolução nos processos de validação ao longo dos últimos anos contribuíram bastante para esta evolução tecnológica da empresa e de sua engenharia de desenvolvimento.

As técnicas foram sendo introduzidas de forma gradativa. E alguns erros no processo de validação foram encontrados. Estes erros ocorreram devido às dificuldades do recém-montado time de Validação de Produto em se adaptar ao novo processo introduzido. E foi justamente por causa desta introdução gradativa, e planejada, que os erros encontrados nos testes anteriores foram corrigidos com a introdução de novas técnicas, que vieram a somar, em termos de qualidade, às técnicas já implementadas. Outra vantagem visível da introdução gradativa de novas técnicas no processo de validação de SW é que a cada novo projeto, ou mesmo a cada nova fase de validação, o processo melhorava, ou seja, a cada novo ciclo de testes o processo de validação era renovado, atualizado, corrigido e consequentemente melhorado. Hoje o processo de validação está em um nível de maturidade bastante elevado e bem consolidado no desenvolvimento de novos produtos.

5.3 CONSIDERAÇÕES FINAIS

As informações apresentadas neste capitulo permitiram destacar aspectos positivos com a inserção de métodos e processos nas validações de software desta empresa do ramo agrícola, bem como as constantes atualizações e evoluções em sua metodologia de testes. Essa inserção, atualizações e evoluções na metodologia utilizada pela área de validação tornaram os produtos da companhia mais confiáveis, deram um grande “know how” à equipe responsável pela verificação de software, diminuição dos períodos de testes e permanência em campo, e a consequente economia de tempo, de recursos e obviamente de gastos. Testes que antes duravam um mês, atualmente, com a aplicação de metodologias de validação de software, duram uma ou duas semanas no máximo. Com isso a empresa passou a economizar em muitos casos, sem perda de qualidade na atividade, até 3/4 do que normalmente era gasto com despesas relacionadas a testes no campo.

Com a introdução de um processo de verificação de SW, situações de bugs críticos (que deveriam ser encontrados nas fases de validação) que nas primeiras validações eram encontrados em fases que antecediam a produção, na produção ou até mesmo após a produção (no campo ou ainda no cliente – situação critica para o produto e para a sua continuidade no mercado) passaram a ser bem raras nos produtos da empresa.

Os resultados finais, como grande parte da metodologia de validação de sistema embarcado aplicada nos processos de validação desta empresa ao longo dos desenvolvimentos de seus inúmeros produtos, de diferentes níveis de complexidade, se revelam bem satisfatórios e muito positivos. Dessa forma, espera-se que o modelo de referência de desenvolvimento de software, a metodologia e os processos de validação de software embarcado (associados ao modelo V) sirvam como referência para processos de melhoria, com o objetivo de tornar desenvolvimentos de software mais rentáveis, ágeis, eficientes e confiáveis.

6 CONCLUSÃO

A maior parte dos defeitos no processo de desenvolvimento de um sistema é de origem humana e são gerados na comunicação, na transformação ou tradução incorreta de informações. Estes defeitos continuam presentes nos produtos de SW e a maioria destes encontra-se em partes do código raramente executadas.

O objetivo do Teste de Sistema é revelar a presença de erros, porém a inexistência de erro raramente é porque o código desenvolvido é de alta qualidade. Geralmente é porque os Casos de Teste são de baixa qualidade. O projeto de casos de teste é um dos melhores mecanismos para a prevenção de defeitos e é tão eficaz em identificar erros quanto à execução dos casos de teste. Um bom Caso de teste tem alta probabilidade de revelar um erro ainda não descoberto.

É interessante sempre ter desenvolvedores experientes porque eles escrevem códigos corretos ou próximos do correto. O teste deve ser conduzido por terceiros, pois testes conduzidos por outras pessoas (que não aquelas que produziram o código) têm maior probabilidade de encontrar defeitos.

A atividade de teste jamais será completada (é impossível executar todas as combinações de caminhos de teste), a carga de testes simplesmente será transferida do projetista ou Engenheiro de Testes para o cliente. Além disso, os testes devem ser planejados bem antes de serem realizados.

Ferramentas de Automatização de Atividades de Teste contribuem para diminuir as falhas produzidas pela intervenção humana, aumentam a qualidade e a produtividade dos testes e ainda aumentam a confiabilidade do programa.

Os resultados mostram também que a construção de modelos de dispositivos de HW é fundamental para sistemas embarcados, e como uma importante ferramenta de auxílio na depuração de códigos. A simples execução dos modelos de dispositivos de hardware já é capaz de eliminar uma série de possibilidades de configurações incorretas e aceleram a detecção de falhas ou bugs durante os ciclos de validação.

É possível finalmente concluir então que quanto mais cedo uma falha, de software for identificado através de testes no código, menor o custo de correção do defeito e maior a probabilidade de corrigi-lo corretamente. E a melhor solução para se chegar a isso é introduzir atividades de Verificação, Validação e Testes ao

longo de todo o ciclo de desenvolvimento. Também é possível observar que quando grande parte dos testes é executada em um ambiente de desenvolvimento de software de aplicação o desenvolvimento é pensado desde o início do projeto visando à qualidade dos testes. Assim é notório que toda a metodologia proposta tem viabilidade prática, já que o tempo de desenvolvimento do programa é reduzido e, quando testado segundo o modelo V, o sistema embarcado apresenta resultados positivos em menos tempo, com redução de custos e com alto grau de qualidade e confiabilidade.

REFERÊNCIAS

BARBOSA, E.; MALDONADO, J.; VINCENZI, A.; DELAMARO, M.; SOUZA, S.; JINO, M. Introdução ao Teste de Software. XIV Simpósio Brasileiro de Engenharia de Software, 2000.

FHWA Homepage. Disponível em: http://ops.fhwa.dot.gov/publications/seitsguide/section3.htm. Acesso em: 2016.

KANG, B.; KWON, Y.; LEE, R. A Design and Test Technique for Embedded Software. In: 3rd Acis Int'l Conference On Software Engineering Research, Management And Applications, SERA. Proceedings Washington: IEEE Computer Society, 2005.

MALDONADO, José. "Teste de Software". In: Qualidade de Software: Teoria e Prática. São Paulo: Prentice Hall, 2001.

MESZAROS, G. XUnit Test Patterns, Refactoring Test Code. 3a ed. Massachusetts, USA: Pearson Education, Inc, 2009.

PC-LINT Homepage. Disponível em: http://gimpel.com/. Acesso em: 2016.

PEZZÉ, M.; YOUNG, M. Teste e Análise de Software Processos Princípios e Técnicas. São Paulo: Editora Artmed SA, 2008.

PFALLER, C. Et al. On the Integration of Design and Test - A Model-Based Approach for Embedded Systems. In: International Workshop On Automation Of Software Test, 2006. Shanghai. Proceedings New York: ACM Press, 2006.

PFLEEGER, S. Engenharia de Software: Teoria e Prática. São Paulo: Prentice Hall, 2ª edição, 2004.

PRESSMAN, Roger. Engenharia de Software. Rio de Janeiro: McGraw Hill, 5ª edição, 2002.

SCITOOLS Homepage. Disponível em: http://www.scitools.com. Acesso em: 2016.

SEO, J. Et al. Which Spot Should I Test for Effective Embedded Software Testing? In: The Second International Conference on Secure System, 2008.

SEO, J.; SUNG, A.; CHOI, B.; KANG, S. Automating Embedded Software Testing on an Emulated Target Board. In: Second International Workshop on Automation of Software Test, Ast. Washington: 2007.

SOMMERVILLE, I. Engenharia de Software. São Paulo: Addison-Wesley, 6ª edição, 2003.

SUNG, A.; CHOI, B.; SIN, S. An interface test model for hardware-dependent software and embedded OS API of embedded system. Proceedings Computer Standard & Interface. 2007.

WIKIWAND Homepage. Disponível em: http://www.wikiwand.com/pt/Modelo_V. Acesso em: 2016.