

# Desenvolvimento de firmware robusto e multiplataforma

Jonathan Gonzaga<sup>1</sup> and Orientador<sup>2</sup>

<sup>1</sup>Graduando em Engenharia de Computação, UNISAL São José - Campinas, [jonathan.s.gonzaga@gmail.com](mailto:jonathan.s.gonzaga@gmail.com)

<sup>2</sup>Professor do UNISAL São José - Campinas, [orientador@sj.unisal.br](mailto:orientador@sj.unisal.br)

**Resumo** - Este artigo .

**Palavras-chave:** sistemas embarcados, firmware, TDD, testes unitários

**Abstract** - This article .

**Keywords:** embedded systems, firmware, TDD, unit tests

## INTRODUÇÃO

Com o avanço da eletrônica e da computação, a miniaturização de circuitos integrados e a redução de custos de fabricação, surgiram na indústria diversos dispositivos eletrônicos com poder de processamento. A maioria desses dispositivos conta com processadores, memórias e periféricos integrados, de forma que seu *software* é destinado e *embarcado* numa aplicação específica. Sistemas dessa natureza são conhecidos como *sistemas embarcados*.

Produtos como eletrodomésticos, eletrônicos em geral, equipamentos médicos, equipamentos de telecomunicações, ferramentas eletrônicas, sistemas de controle e automação e sistemas de tempo real em veículos são ou possuem sistemas embarcados em sua concepção.

Devido ao alto nível de criticidade de alguns sistemas embarcados, é esperado que o *software* executado neles seja altamente confiável e com o mínimo de *bugs* possível. A realidade da indústria de eletrônicos mostra que essa afirmação nem sempre é verdadeira.

Erros, falhas ou *bugs* de *software* são um problema em diversas áreas da tecnologia. Desde *bugs* que facilitam a ação de *hackers* em redes sociais, *bugs* em *smartphones* que causam travamento do sistema operacional a *bugs* em sistemas de freios que causam acidentes em veículos. Esse último caso é ainda mais grave, pois o sistema lida diretamente com vidas humanas.

Especialistas em desenvolvimento de *software* embarcado, como Jack Ganssle, James W. Grenning e Jacob Beningo, dedicaram-se a criar literatura de qualidade, escrevendo livros e artigos com técnicas e metodologias de desenvolvimento de *software*. Alguns títulos relevantes são: *Test-Driven development for Embedded C* [1], *The Art of Designing Embedded Systems* [2], *Reusable Firmware Development* [3] entre outros. A partir dessas obras muito se tem discutido sobre como criar sistemas mais seguros, com menos *bugs* e menos suscetíveis a erros do usuário.

No processo de desenvolvimento de *software*, o custo de uma alteração no código tende a ficar maior conforme as etapas do desenvolvimento avançam, por isso o custo de um *bug* encontrado em campo após o produto ser lançado, é muito maior que o custo do mesmo *bug* sendo encontrado ainda na fase de desenvolvimento [4]. Esse cenário por si só já mostra a necessidade de se detectar problemas nas fases iniciais do projeto.

*Software* embarcado muitas vezes é difícil de ser testado e validado, extremamente dependente da plataforma de *hardware target*, e tende a demonstrar problemas de integração com outras partes do sistema após a inserção de novas funcionalidades. Devido a limitação de recursos e a extrema dependência do *hardware*, testes automatizados não são realizados, o que acaba prejudicando a qualidade do código.

Metodologias e conceitos como *TDD* (*Test-Driven development*), pirâmide de testes, *SOLID* e outros, são historicamente e erroneamente atribuídos apenas aos *softwares* de "alto nível", como *web* ou *mobile* por exemplo, afastando os desenvolvedores de *software* embarcado desses conceitos e da utilização de abstrações mais inteligentes.

Com essas dificuldades em mente, percebe-se a necessidade de se estimular as boas práticas de desenvolvimento de *software* embarcado, a utilização de testes automatizados e independentes do *hardware*, e o estudo de conceitos de engenharia de *software* que auxiliem na redução de *bugs* e consequentemente, redução de custos do projeto e aumento da qualidade do código gerado.

## REFERENCIAL TEÓRICO

### A. Firmware.

*Firmware* é um tipo específico de *software* executado diretamente num circuito integrado (ou *chip*). Não necessita de outros programas para ser executado (como sistemas operacionais), além de servir a um propósito único. Em outras palavras, *firmware* é o *software* executado em um sistema embarcado. [5]

Devido a limitações de tamanho e recursos desse tipo de sistema, o *firmware* precisa manipular o *hardware* diretamente e toda a sua arquitetura costuma ser voltada a eventos do mundo externo.

Diferente de sistemas computacionais mais complexos e com alto nível de abstração, onde geralmente o *kernel* do sistema operacional é modularizado e abstrai o acesso a dispositivos de *hardware*, num sistema embarcado o *firmware* é responsável pela gerência dos recursos e eventos de *hardware* (interrupções e excessões do processador) e também pelo código da aplicação (regras de negócio, interface com usuário e demais especificidades).

Muitas vezes o *firmware* faz parte de um sistema computacional maior, por exemplo, computadores *desktop* possuem circuitos integrados para aplicações específicas, como a *BIOS* (*Basic Input/Output System*).

Algumas literaturas não fazem distinção entres os termos *firmware* e *software embarcado*, o que pode gerar certa confusão: *firmware* é o *software* executado num circuito integrado comumente escrito em linguagem C, C++, ou mesmo *Assembly*. Já o *software embarcado* possui característi-

cas de alta abstração, sendo uma aplicação (um processo rodando num sistema operacional, geralmente baseado em GNU/Linux) porém executado em dispositivos de propósito específico (roteadores, terminais de auto atendimento). Pode ser escrito em linguagens de programação como C, C++, Rust, Go, Python entre outras.

Num sistema embarcado executando um *firmware*, o cérebro por trás de todo o processamento é um componente chamado *microcontrolador*.

#### B. Microcontroladores.

Microcontroladores são processadores de pequeno porte com CPU, memórias RAM, ROM, FLASH e recursos atrelados no mesmo encapsulamento ou SoC (*System on Chip*). Esses recursos denominados periféricos, são inclusos no *chip* com o intuito de tornar possível o interfaceamento entre a CPU e o mundo externo através dos pinos físicos do componente.

Os recursos mais comuns em microcontroladores são as interfaces de comunicação serial (USART, I2C, SPI), conversores analógico-digital e digital-analógico (ADC e DAC), temporizadores/contadores (TIMERS) e demais componentes.

Microcontroladores são usados em produtos e dispositivos automatizados, como os sistemas de controle de automóvel, dispositivos médicos implantáveis, controles remotos, máquinas de escritório, eletrodomésticos, ferramentas elétricas, brinquedos e outros sistemas embarcados.

Ao reduzir o tamanho e o custo em comparação a um projeto que usa um dispositivo microprocessado, microcontroladores tornam-se econômicos para controlar digitalmente dispositivos e processos. [6]

#### C. Testes de software.

Testes de *software* são procedimentos pelos quais o código fonte de um *software* é submetido afim de validar seu comportamento mediante os requisitos pelos quais foi projetado. Existem diversos tipos de testes de *software*, cada um responsável por testar partes e situações diferentes. Podemos citar:

- Testes *end-to-end*
- Testes de integração
- Testes unitários

#### D. Testes unitários.

Testes unitários são a base dos testes de *software*. Na pirâmide de testes, encontram-se no nível mais baixo, o que indica que são o tipo mais barato, mas também os mais fáceis de se implementar. [7]

Como seu nome sugere, são testes de *unidade*, onde é possível testar a menor unidade testável do código (sejam funções, classes ou módulos).

#### E. TDD - Test-Driven development.

*Test-Driven development* (ou *Desenvolvimento orientado a testes*) é uma técnica incremental de construção de *software*. Nessa técnica, nenhum código de produção é escrito sem que primeiro seja escrito um teste unitário que falhe na primeira execução.

Ao contrário da prática comum de desenvolvimento de *software*, onde primeiro é desenvolvido o código de produção e só depois os testes, no TDD o desenvolvedor expressa o comportamento desejado do código em um teste. O teste é executado e falha. Só então ele escreve o código de produção, fazendo o teste passar. A automação de testes é a chave para o TDD. Os testes são pequenos e automatizados. A cada nova funcionalidade implementada, novos testes unitários são escritos, seguidos imediatamente por um código de produção que satisfaça aqueles testes.

Conforme o código de produção cresce, também crescem em conjunto os testes unitários, que são ativos tão valiosos quanto o próprio código de produção. A cada mudança de código, o conjunto de testes é executado, verificando a funcionalidade da nova implementação, mas também a compatibilidade com o código já existente. [1]

### MATERIAIS E MÉTODOS

#### A. Software - Ceedling.

*Ceedling* é um *build system* (sistema de construção de *software*) para projetos escritos em linguagem C. É uma espécie de extensão do sistema de construção *Rake* (*make-ish*) do Ruby. O *Ceedling* é voltado principalmente para o Desenvolvimento Orientado a Testes (TDD) em C e é projetado para reunir as ferramentas CMock, Unity e CException - três outros projetos de código aberto que auxiliam na dinâmica de testes automatizados. [8]

#### B. Software - FreeRTOS.

*FreeRTOS* é um *kernel* de tempo real, ou um sistema operacional de tempo real (*Real-Time Operating System*) para dispositivos embarcados. Foi desenvolvido para ser pequeno, simples e portátil. Seu *kernel* é composto por apenas 3 arquivos em linguagem C. O FreeRTOS permite a fácil implementação de multitarefa preemptiva ou não preemptiva com diversos níveis de prioridade de tarefas. [9]

#### C. Arquitetura de software do sistema.

Abaixo pode-se ver os componentes de *software* separados por nível de abstração:

### METODOLOGIA

A metodologia utilizada foi

### DESENVOLVIMENTO

O processo de desenvolvimento do projeto seguiu o padrão...

#### A. Dual-targeting: compilando para duas arquiteturas.

Um dos pilares do desenvolvimento de *software* multiplataforma é a abstração. Utilizar interfaces bem definidas a partir de *header files*, variando apenas a implementação dessas interfaces nos *source files*.

#### B. Camadas de abstração - drivers e RTOS.

Para se escrever código portátil é extremamente necessário a não dependência de *hardware*, RTOS - (*Real Time Operating Systems*) e demais bibliotecas

### C. Testes unitários.

Para escrever testes unitários, é necessário a utilização de um *framework*

## CONCLUSÃO

Conclui-se que

## Referências

- 1 GRENNING, J. W. *Test Driven Development for Embedded C*. [S.l.]: Pragmatic bookshelf, 2011.
- 2 GANSSLE, J. *The art of designing embedded systems*. [S.l.]: Newnes, 2008.
- 3 BENINGO, J.; BENINGO, J.; ANGLIN. *Reusable Firmware Development*. [S.l.]: Springer, 2017.
- 4 EMBARCADOS. Editorial: custo do firmware. *Embarcados*, Embarcados, 2015. Disponível em: <https://www.embarcados.com.br/editorial-custo-do-firmware/>.
- 5 GANSSLE, J. *The firmware handbook*. [S.l.]: Elsevier, 2004.
- 6 GRIDLING, G.; WEISS, B. Introduction to microcontrollers. *Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group*, 2007.
- 7 CONTAN, A.; DEHELEAN, C.; MICLEA, L. *Test automation pyramid from theory to practice*. [S.l.: s.n.], 2018.
- 8 GOMES, E. C.; AMORA, P. R.; TEIXEIRA, E. M.; LIMA, A. G.; BRITO, F. T.; CIOCARI, J. F.; MACHADO, J. C. Utos: A tool for testing uefi code in os environment. In: SPRINGER. *IFIP International Conference on Testing Software and Systems*. [S.l.], 2016. p. 218–224.
- 9 ZHU, M.-Y. Understanding freertos: A requirement analysis. *CoreTek Syst., Inc., Beijing, China, Tech. Rep*, 2016.
- 10 MARTIN, R. C. *Clean code: a handbook of agile software craftsmanship*. [S.l.]: Pearson Education, 2009.
- 11 MARTIN, R. C. *Clean architecture: a craftsman's guide to software structure and design*. [S.l.]: Prentice Hall, 2018.
- 12 DENARDIN, G. W.; BARRIQUELLO, C. H. *Sistemas operacionais de tempo real e sua aplicação em sistemas embarcados*. [S.l.]: Editora Blucher, 2019.

[10] [11] [12]