UNIVERSIDADE PAULISTA – UNIP

JONATHAN JONYS MIRANDA, C4400D-2

BOAS PRÁTICAS DE PROGRAMAÇÃO UTILIZANDO 5S

LIMEIRA
2018

JONATHAN JONYS MIRANDA

BOAS PRÁTICAS DE PROGRAMAÇÃO UTILIZANDO 5S

Trabalho de conclusão de curso para obtenção do título de graduação em Ciência da Computação apresentado à Universidade Paulista – UNIP.

Orientador: Prof. Me. Antonio Mateus Locci
Prof. Me. Marcos Vinicius Gialdi
Prof. Me. Sérgio Eduardo Nunes
Prof. Me. Sandra Maria Crippa

LIMEIRA
2018

JONATHAN JONYS MIRANDA

BOAS PRÁTICAS DE PROGRAMAÇÃO UTILIZANDO 5S

Trabalho de conclusão de curso para obtenção do título de graduação em Ciência da Computação apresentado à Universidade Paulista – UNIP.

Aprovado em:

BANCA EXAMINADORA

_____ ___/___/____
Nome do Membro da Banca
Instituição do Membro da Banca

_____ ___/___/____
Nome do Membro da Banca
Instituição do Membro da Banca

_____ ___/___/____
Nome do Membro da Banca
Instituição do Membro da Banca

Dedico este trabalho primeiramente a minha mãe Deise Zia Miranda, que me acompanhou desde o início e me apoiou no decorrer de toda minha graduação.

Ao meu pai Edemir Antônio Miranda, já falecido, por sempre me encorajar, acreditar em mim e por me incentivar a fazer tudo o que eu desejava.

Aos meus amigos Bruno de Almeida Rossetto, Gabrielli Elias Martins, Mayara Cristina Soldan e Oliver Silva de Oliveira, que me acompanharam desde o começo, me acalmaram durante minhas frustrações e me ofereceram apoio durante o decorrer deste trabalho.

AGRADECIMENTOS

Agradeço primeiramente a Deus pelo intelecto fornecido e por ser à base de minhas conquistas.

A minha mãe Deise Zia Miranda por acreditar em minhas escolhas, me apoiando e se esforçando junto a mim para que eu atingisse meus objetivos.

Aos professores Amaury Bosso Andre, Antonio Mateus Locci, Fabio Roberto Sciamana, Herbert Adriano Stabile, João Augusto Cardoso, Marcos Henrique Pitoli, Marcos Vinicius Gialdi, Paulo Estevão Bueno de Camargo e Sérgio Eduardo Nunes por me acompanharem ao decorrer da minha graduação ajudando de todas as formas possíveis para que eu atingisse meus objetivos.

A empresa PIUMA – Soluções em Tecnologia e seus funcionários Gabriel Felipe Kugel, Lucas Tafarel Baquião dos Santos, Otavio Candido de Araujo Neto e Renato Silvestre da Silva pelo auxilio no desenvolvimento do trabalho.

“Você não pode impor a produtividade, você deve fornecer as ferramentas para permitir que as pessoas se transformem no seu melhor.”

(Steve Jobs)

RESUMO

Atualmente o modelo de negócio empresarial, em grande parte das empresas, é voltado para o 5S, já que é a base para implantação da qualidade total, pois mobiliza, motiva e conscientiza a empresa através de organização e disciplina. Devido ao seu grande sucesso nestes ambientes a ideia surgiu de uma premissa de que pode ser empregado no ambiente de desenvolvimento de códigos, com a finalidade de organizar a atmosfera para se criar códigos mais compreensíveis. Considerando que hoje em dia os programadores criam cada vez mais códigos sob pressão e estresse devido à alta demanda, foram compilados diversos conceitos se baseando em modelo 5S para boas práticas de programação. Através da necessidade de uma organização no ambiente de desenvolvedores, surgiu a iniciativa de utilizar um método de organização eficaz para evitar desmotivação e melhorar os padrões de estrutura de código e elevar a produtividade. Assim, os códigos desenvolvidos pelos programadores seriam mais limpos visualmente, e seria mais fácil prestar uma manutenção, caso haja a necessidade, pois um código que segue um padrão de boas práticas será facilmente compreendido por qualquer programador que compreenda a linguagem. Utilizando uma pesquisa exploratória e uma abordagem qualitativa, o modelo de qualidade 5S, que é uma cultura organizacional, nesta pesquisa, voltada para a área computacional, visa a melhoria continua de codificação, propondo boas práticas para garantir a qualidade total, fortificada através de análise de documentos e pesquisas bibliográficas. Com a utilização deste método se espera detectar melhorias contínuas no processo de codificação, facilidade maior de compreensão e fácil manutenção em códigos, garantindo a melhoria de desempenho do trabalho exercido. Em relevância dos aspectos analisados na qualidade da cultura organizacional empregada no 5S, o método se comprova eficaz em grande parte das implantações, sendo uma ótima alternativa para organização dos padrões estruturais de empresas e programadores que trabalham com desenvolvimento de código.

Palavras chave: 5S. Boas práticas. Programas. Programadores.

ABSTRACT

Currently the business model, in most of the companies, is focused on 5S, since it is the basis for implementation of the total quality, as it mobilizes, motivates and makes the company aware through organization and discipline. Due to its great success in these environments the idea arose from a premise that it can be used in the code development environment, in order to organize the atmosphere to create more understandable codes. Considering that nowadays programmers are creating more and more code under pressure and stress due to high demand, several concepts have been compiled based on 5S model for good programming practices. Through the need for an organization in the developer environment, the initiative has emerged to use an effective organization method to avoid demotivation and improve code structure standards and increase productivity. Thus, code developed by programmers would be visually cleaner, and maintenance would be easier if needed, because a code that follows a standard of good practice will be easily understood by any programmer who understands the language. Using an exploratory research and a qualitative approach, the 5S quality model, which is an organizational culture, in this research, focused on the computational area, aims at the continuous improvement of coding, proposing good practices to guarantee the total quality, fortified through analysis of documents and bibliographic research. The use of this method is expected to detect continuous improvements in the coding process, greater ease of understanding and easy maintenance in codes, guaranteeing the improvement of performance of the work performed. In terms of relevance of the aspects analyzed in the quality of the organizational culture used in 5S, the method proves to be effective in most of the implementations, being a great alternative for the organization of the structural patterns of companies and programmers working with code development.

Keywords: 5S. Good habits. Software. Programmers.

SUMÁRIO

1 INTRODUÇÃO 9
1.1 Objetivo 9
1.2 Justificativa 10
1.3 Metodologia 10
2 CONVERTENDO OS 5S 11
2.1 Utilização (Seiri) 11
2.1.1 Componentização 11
2.1.2 Responsabilidade Única 12
2.2 Organização (Seiton) 13
2.2.1 Design Patterns 13
2.2.1.1 KISS – Mantenha Isto Estupidamente Simples 14
2.2.1.2 DRY – Não se repita 15
2.2.1.3 TDA – Fale, Não Pergunte 15
2.2.1.4 YAGNI – Você Não Vai Precisar Disso 16
2.2.1.5 SOC – Separação de Responsabilidades 16
2.3 Limpeza (Seisou) 17
2.3.1 Código Limpo 17
2.3.2 Sala Limpa 18
2.4 Saúde e Higiene (Seiketsu) 19
2.4.1 Pomodoro 19
2.4.1.1 Gerenciamento de Tempo 20
2.4.1.2 Beneficios do Pomodoro 20
2.4.2 Ergonomia 21
2.4.2.1 LER – Lesão Por Esforço Repetitivo 22
2.5 Auto Disciplina (Shitsuke) 22
2.5.1 Gerenciamento de Projeto 22
2.5.2 Versionamento 23
2.5.3 Scrum 24
2.5.4 Equipes Auto Gerenciadas 25
2.5.5 Pontos Por Função 25

3 HIPÓTESE.....26
CONSIDERAÇÕES FINAIS 28
REFERÊNCIAS BIBLIOGRÁFICAS 29

1 INTRODUÇÃO

Toda empresa tem que se organizar corretamente para que o seu processo de produção seja bem executado dentro do prazo, das normas e com qualidade. Na programação não é diferente, para que o código seja bem escrito e compreendido é necessário se utilizar boas práticas de programação, deste modo o *software* estará mais bem documentado, claro e coeso com suas funções.

Os 5S foram criados no Japão, primordialmente para garantir um ambiente adequado para uma maior produtividade, sendo que grandes organizações afirmam que os 5S é a base comportamental e física para o sucesso das ferramentas utilizadas na gestão.

Neste caso, a alternativa sugerida seria utilizar este modelo atualmente empregado em empresas de produção, e aplica-lo diretamente nos programadores de *softwares*, utilizando boas práticas empregando os 5S. A definição de cada etapa é: utilização (Seiri) que visa utilização de recursos, organização (Seiton) que visa a estrutura do código, limpeza (Seiso) representa os padrões de limpeza do código, saúde e higiene (Seiketsu) que é mais voltado para o programador em sua saúde mental, física e social, e a auto disciplina (Shitsuke) que remete a aplicar corretamente todas as outras etapas. Os 5S adaptado para programadores, tende a garantir a qualidade de serviço utilizando as boas práticas, de modo que cada etapa seja executada corretamente, garanta a qualidade total do seu serviço.

1.1 Objetivo

O principal objetivo do trabalho é ajudar os programadores, de modo que ao exercer o trabalho, seja garantido um trabalho coeso e claro, facilitando assim a compreensão para conferencias posteriores com a visualização de um código organizado e limpo. Porém, também ajudar as empresas de *software*, para que haja um aumento de produtividade.

1.2 Justificativa

Atualmente existem diversos modelos empregados em computação voltados para organização. Contudo, devido a grande eficácia comprovada em empresas que utilizam os 5S e verificando a necessidade de uma estrutura melhor empregada para programadores, surgiu-se a ideia de adaptar o modelo dos 5S.

1.3 Metodologia

A metodologia deste método funciona em ciclos contínuos, já que um completa o outro, todos acontecem simultaneamente de modo que tudo se resume como se fosse um único processo. Através de boas práticas já reconhecidas em modelos bibliográficos que estão compiladas nesta pesquisa, é possível garantir que a qualidade total do código será atingida baseando-se em aplicações já empregadas dos 5S.

Imagem 1 - Ciclo dos 5S

Fonte: Jonathan Jonys Miranda

2 CONVERTENDO OS 5S

Os 5S surgiram previamente onde a utilização, organização, limpeza, saúde e higiene e a auto disciplina são utilizados em ambientes de produção industrial para garantia de qualidade. Abaixo os 5S serão empregados no ambiente de desenvolvimento de *software* utilizando os 5S em um eixo computacional, voltado para empresas de *software* e programadores.

2.1 Utilização (Seiri)

A utilização se refere ao senso de utilização dos recursos utilizados no código. A ideia é separar tudo que não é útil e descartar, e o que for útil ser melhorado. Evitando um código carregado de informações desnecessárias e garantindo sempre que o código esteja atualizado e funcionando corretamente com as ferramentas mais adequadas.

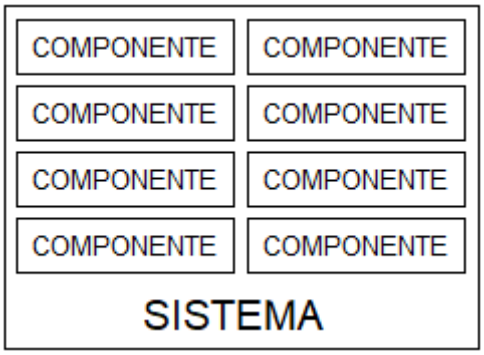
2.1.1 Componentização

O desenvolvimento por componentização é arquitetural, baseado em divisão de sistemas em unidades menores. É só tratar o sistema como minissistemas conjuntos, diminuindo a complexidade e permitindo o foco somente na funcionalidade daquele módulo ou no conjunto de módulos semelhantes, podendo assim ser reutilizadas em diversas outras aplicações através do acesso aos componentes. Desde modo é possível garantir que ele será fracamente acoplado, facilitando a manutenção do módulo se for necessário e evitando erros graves.

Este tipo de programação garante uma economia muito grande de tempo de desenvolvimento, principalmente em longo prazo, considerando o portfólio de componentes. O padrão de desenvolvimento também é um fator positivo, já que a equipe constrói em moldes para que sejam sempre integrados a novas aplicações. Sem falar é claro, da maior robustez e qualidade, já que todos foram fortemente testados.

A imagem 2 mostra visualmente como seria realizada uma componentização.

Imagem 2 - Componentização



Fonte: Jonathan Jonys Miranda

2.1.2 Responsabilidade Única

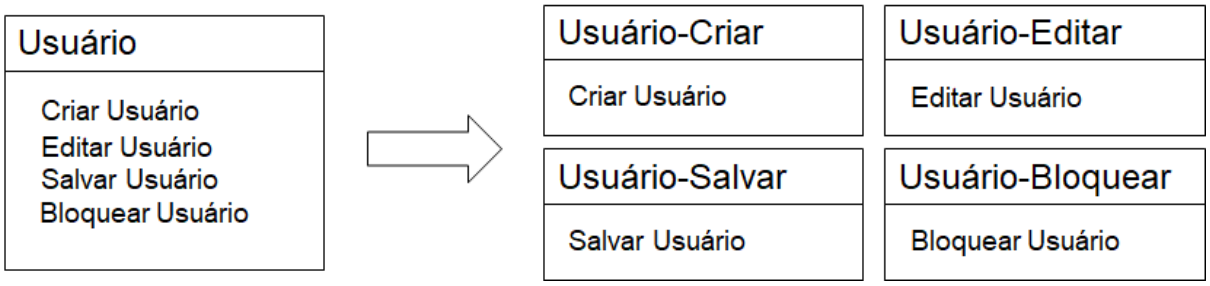
A responsabilidade única assume que o projeto a ser realizado se baseia em que a classe responsável por realizar ações assume mais coisas do que deveria assumir. Esses tipos de classes são conhecidas como “canivete suíço” já que são multifuncionalidades. Quando se trata de programação, isso não é algo positivo, pois encontrar um erro no meio de diversas ações que são realizadas pela classe podem desencadear somente mais falhas.

A melhor prática que pode ser realizada utilizando responsabilidade única é onde cada classe realiza somente o que ela deve fazer. De forma desacoplada cada classe realiza somente seu papel, herdando entidades e possuindo propriedades comuns.

Nesta boa prática entende-se que se existe mais de um motivo para ser alterada, ela não segue esta prática. Aplicar este princípio de responsabilidade única é importante para uma arquitetura madura e sustentável.

A imagem 3 representa visualmente como seria realizado o principio de responsabilidade única.

Imagem 3 - Responsabilidade Única



Fonte: Jonathan Jonys Miranda

2.2 Organização (Seiton)

A organização envolve a importância da disponibilidade e acesso imediato. O principio é que tudo tem seu lugar certo e a ordem deve sempre ser mantida para evitar a perda desnecessária de tempo. Se tudo estiver organizado corretamente, será garantido que o código funcionará como se deve.

2.2.1 Design Patterns

Os padrões de projeto, ou como são conhecidos, *design patterns*, surgiram a partir da motivação de ajudar a solucionar problemas frequentes, de modo que se usadas corretamente, se tornam ferramentas muito importantes para desenvolvedores. Esta prática são soluções de templates de alto nível para soluções e não como uma solução em sí.

Os *design patterns* foram organizados em três grupos. Os padrões de criação se referem a construção do objeto e o de referência. Os padrões estruturais se tratam da interação entre objetos que foram grandes e complexos objetos. E os padrões comportamentais, que cuidam especificamente da responsabilidade e dos algoritmos.

Esta prática tem um grande peso, pois são soluções que já foram testadas por pessoas que já a utilizaram, dando a confiança em relação a sua eficácia. Esse conceito é muito focado em reutilização de soluções, porém entendendo que os

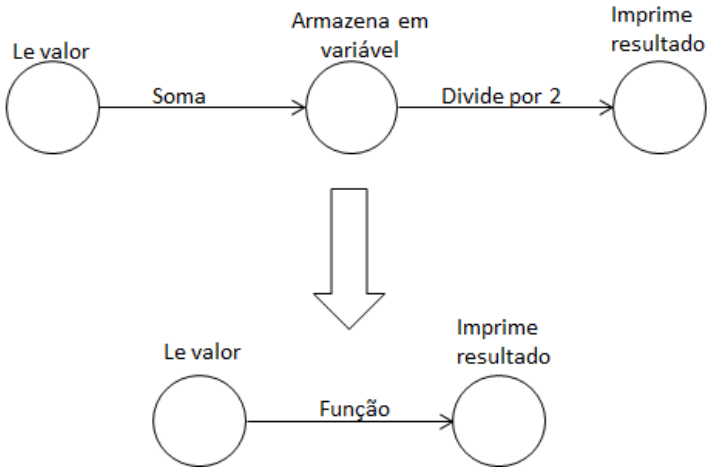
problemas não são iguais, é necessário desmembrar e encontrar similaridades, deste modo é possível encontrar possíveis soluções.

Atualmente existem diversos princípios de *design* que se tornaram boas práticas através dos anos, ajudando a formar diversos princípios como KISS que é focado em simplificar as soluções evitando a complexidade desnecessária. DRY que foca em evitar a repetição de qualquer parte do sistema, seja em código ou até mesmo lógica duplicada. *Tell, Don't Ask*, que se refere à atribuição de responsabilidades para as classes de forma correta. YAGNI que surge da premissa da necessidade das funcionalidades, para que não se programe algo que vai ser deixado de lado por não ser uma necessidade. SoC, que é o processo de dissecar o *software* em características diferentes, compactando um comportamento e dados que podem ser utilizados em outras classes.

2.2.1.1 KISS – Mantenha Isto Estupidamente Simples

A ideia dessa prática de *design pattern* é manter o código simples, utilizando as soluções mais simples para resolver o problema, deste modo evitaria a complexidade desnecessária.

Imagem 4 - Exemplo de Simplicidade



Fonte: Jonathan Jonys Miranda

2.2.1.2 DRY – Não se repita

Esta prática de *design pattern* é manter evitar a repetição de qualquer parte do sistema, a ideia é tornar o sistema o mais simples possível, evitando que haja excesso de trabalho. Esta prática serve tanto para o código que está sendo feito pelo programador, quanto à lógica que é utilizada para a realização.

Na Imagem 5 é possível observar que foi repetido o uma conta que não era necessária.

Imagem 5 - Exemplo de repetição

```
int i;
int j;
int a = i * i;
int b = i * j;
int c;
int d;

c = a + b;

d = i * j;
print d;
```

Fonte: Jonathan Jonys Miranda

2.2.1.3 TDA – Fale, Não Pergunte

Este tipo de *design pattern* serve para que se atribua corretamente as responsabilidades para a classe que se está criando, de modo que o objeto sempre deve realizar a ação, e não perguntar o que deve ser feito. Esta técnica é utilizada para evitar o forte acoplamento no sistema.

2.2.1.3.1 Baixo Acoplamento

O baixo acoplamento em um *software* é uma boa prática muito essencial para evitar problemas complexos, já que quanto mais dependente for o código, mais difícil será encontrar algum problema caso haja uma falha.

Um sistema fortemente acoplado é onde existe um paralelismo de processamento, compartilhamento de memória e periféricos.

Para desacoplar códigos é necessário realizar um *design* de desenvolvimento mais flexível para realização de mudanças. Uma prática que pode ser utilizada é realizar a programação através da interface, permitindo realizar a troca de componentes em tempo de execução.

2.2.1.4 YAGNI – Você Não Vai Precisar Disso

Este principio se refere às funcionalidades necessárias para o sistema, a premissa é evitar que seja adicionado funcionalidades desnecessárias no sistema cujo o programador considere necessário, quando na verdade não é.

Na imagem 6, é possível verificar o ciclo do YAGNI. Sempre é necessário verificar a necessidade de se realizar qualquer coisa no sistema, evitando trabalho em excesso.

Imagem 6 - YAGNI

Fonte: Jonathan Jonys Miranda

2.2.1.5 SOC – Separação de Responsabilidades

Se refere ao processo de dissecar um componente em características responsáveis pelo comportamento e dados que podem ser utilizados em outros classes. Normalmente, é representada por características ou comportamento da

classe. Esta separação aumenta a reutilização de código, melhora a manutenção quando necessário e facilita os testes que serão realizados.

2.3 Limpeza (Seisou)

A limpeza representa a atenção ao ambiente de codificação em todos os aspectos, melhorando a ilegibilidade de código e padronizando para que haja uma melhor compreensão do que está sendo desenvolvido pelo programador.

2.3.1 Código Limpo

A técnica de código limpo foi elaborada para evitar a ilegibilidade do código, permitindo que a codificação fosse aprimorada para que sempre seja compreensível. Na realidade, é um conjunto de práticas que o programador deve adotar, contudo, as diretrizes de código limpo é muito mais do que simplesmente estudar sobre ele, é necessário muita prática, raciocínio e muita determinação para se produzir algo de qualidade.

A criação de um código limpo raramente ocorre na primeira tentativa, porém é de extrema importância que o programador comece a fazer a codificação em cima das diretrizes do código limpo. De inicio é normal que conforme o código toma sua forma, as limpezas comessem a ser realizadas, porém com o tempo, a limpeza do código ocorre de forma natural. Porém é necessário que haja uma auto disciplina do programador, pois caso o código esteja ruim, é necessário que o programador corrija de imediato suas imperfeições, pois a limpeza deve ser feita aos poucos e não somente quando finalizar o código.

No código limpo o código deve sempre ser tratado como uma atividade de risco, o código que estiver mal escrito, é muito difícil de ser corrigido, demanda muito tempo e isso pode afetar os custos do projeto, e em alguns casos pode afetar seu desempenho e funcionamento.

Nesta técnica, a capacidade de visualizar um código mal escrito e imediatamente pensar em uma forma de se limpar vai aumentando com o decorrer do tempo, esta prática se chama sensibilidade de código, onde se pode renomear variáveis, desfazer blocos aninhados, refinar condições, extrair métodos e até mesmo considerar alternativas mais viáveis para tratativa de regras de negocio.

Uma outra boa prática de codificação é fazer o código pensando na leitura, imaginando que outro desenvolvedor futuramente irá interpretá-lo. Antes de implementar os métodos o programador deve medir o nível de legibilidade do mesmo. Neste caso, é necessário se colocar no lugar de outro programador, e encontrar as dificuldades em ler seu próprio código, isso gera a qualidade do seu próprio código, pois quanto mais legível, mais bem escrito o código está.

2.3.2 Sala Limpa

O conceito de sala limpa empregado para qualidade de *software* é muito difundida no desenvolvimento de projetos corporativos. O processo se baseia no projeto apurado das funções, que são analisadas pelo método de revisão com o objetivo de verificar se fazem realmente o que foram especificadas a fazer. O desenvolvimento da sala limpa remove então, a necessidade de depuração do programa, assegurando assim que os erros nunca começam introduzidos no sistema.

Existe outro uso para o termo na tecnologia de programação, que é o uso de equipes separadas para especificação de exigências, implementação de funcionalidades e comparação. A comunicação entre os grupos é realizada geralmente via documentos escritos. Este método então é usado para desenvolver *software* de uma maneira que possa ser demonstrado que não houve infração de questões legais por parte dos desenvolvedores.

Em outras palavras, para qualidade de *software*, o método sala limpa é uma adaptação de um modelo já existente, e sua abrangência inclui a área de *softwares*, que é responsável por documentar e regularizar projetos com a premissa de que nada está bagunçado, com falhas ou bugs em sua criação, então os testes são

realizados para verificar somente o que deve estar funcionado corretamente conforme descrito em sua concepção na regra de negócio da empresa. Deste modo o produto entregue será sempre o que foi prometido ao cliente. Ou seja, o ambiente sala limpa, que é o ambiente controlado da qualidade de *software*, é a área de desenvolvimento de cada programador para realização de testes para padronização do serviço conforme especificações da regra de negócio de um projeto de *software*.

2.4 Saúde e Higiene (Seiketsu)

A saúde e higiene vem de todos os aspectos voltados a saúde física, mental e social do programador. Este é um dos fatores mais importantes, pois afeta diretamente no código

2.4.1 Pomodoro

Um dos maiores problemas para pessoas que trabalham com foco em uma atividade é a dispersão causada por estresse devido ao excesso de tempo de execução. Este desgaste gera uma perda de produção gradativa até o momento em que o mesmo pode acabar não conseguindo mais realizar a codificação. Isso ocorre muito com os desenvolvedores de *software*, mas também com muitas pessoas que trabalham frente ao computador. A melhor forma de continuar a realizar o seu trabalho, seria dando um tempo dele e tranquilizar a mente antes de prosseguir com o que estava fazendo, evitando que o código seja uma abstração do estresse do programador.

O problema do método de distração em geral, é que com o decorrer do tempo esses hábitos de distração podem vir a se tornar mais decorrentes e afetar o desempenho do trabalho, e no fim do dia, a sensação é que não foi possível realizar o que deveria ter sido feito devido à dispersão.

A técnica pomadoro surgiu exatamente para tratar o tempo para que seja possível relaxar a mente, sem que a dispersão afete seu trabalho. A técnica consiste em fracionar o tempo de trabalho com foco total, sem distrações e interrupções, e o

tempo para descanso. Contudo, é muito difícil fazer isso, pois o cérebro sempre vai optar pela distração, para que o método funcione corretamente, é necessário ter a auto disciplina.

Inicialmente a técnica pomodoro funciona com a concepção de trabalho focado durante vinte cinco minutos contínuos, com um descanso de cinco minutos para descanso. Este momento de pausa pode ser qualquer distração, tomar um café, beber água, ir ao banheiro, verificar e-mails, acessar o facebook, entre outras ações. Este trecho de tempo é chamado de um pomodoro. A técnica consiste em quatro pomodoros seguidos e ao finalizar o quarto pomodoro, a pausa se torna maior, sendo ela de quinze minutos. Após este descanso maior, reinicie os pomodoros e faça isso sucessivamente.

2.4.1.1 Gerenciamento de Tempo

O pomodoro não tem o foco somente em foco contínuo e descanso. Todo o trabalho pode ser baseado nesta técnica, de modo que além de ganho de produtividade, é possível gerenciar melhor o tempo do dia a dia.

Por diversas vezes ocorre de que a solução dos problemas surge quando não estamos pensando nele. Existem diversos relatos de pessoas que alegam que as soluções dos problemas surgem ao estar tomando banho ou ao acordar no meio da noite. A técnica surgiu destes momentos, pois a ideia é que nos momentos de descanso do pomodoro que as soluções surjam, pois a mente desliga do problema.

2.4.1.2 Beneficios do Pomodoro

O pomodoro trás diversos benefícios para quem a aplica corretamente como alivio de ansiedade, já que com as pausas, aquela sensação de que falta pouco para terminar diminui. Aumenta o foco e a concentração, já que você se força a isso, para que haja melhor desempenho na realização do trabalho. Diminui as distrações e as interrupções, pois normalmente quando elas ocorrem é quando o programador não consegue mais programar e até que se desligue do problema demora muito tempo,

e com o foco e a concentração empregados durante o desenvolvimento, fazem com que os mesmos não dispersem ao estar realizando a tarefa. O aumento da motivação e determinação em busca de objetivos se tornam mais frequentes, pois não há um estresse ao executar as tarefas, pois regularmente haverá um tempo de descanso. Entre diversos outros benefícios como assertividade em estimativas, habito de concentração, melhora no processo de trabalho e estudo.

2.4.2 Ergonomia

O objetivo da ergonomia é a ciência que estuda as adaptações do posto de trabalho, neste trabalho, relacionada a padrões comportamentais ao programador frente ao computador. Os padrões são normatizados pelas normas regulamentadoras, que devem ser sempre verificadas para que os funcionários sempre às cumpram, evitando problemas graves relacionados a má postura.

Imagem 7 – Postura adequada conforme normas regulamentadoras

Fonte: Oficina da Net

2.4.2.1 LER – Lesão Por Esforço Repetitivo

Um dos problemas relacionados à má relacionados a uma má postura é a lesão por esforço repetitivo, muito comum em pessoas que trabalham frente ao computador, entendida como doença ocupacional.

Esta doença aparece lentamente no ser humano, e muitas vezes nem é notada ao longo de uma vida de trabalho, ou quando é evidente, já ocorreu um comprometimento grave com a área afetada.

Existem medidas preventivas para evitar a LER, como pausas periódicas em trabalhos de digitação, levantar periodicamente para movimentar-se, obter postura adequada (pulsos sem torção, ombros não tensionados, costas apoiadas no encosto da cadeira, pés encostando no chão ou apoio de pés, entre outros). Essas medidas são muito importantes para evitar lesões.

2.5 Auto Disciplina (Shitsuke)

A auto disciplina tem o intuito de atenção e auto gestão, de modo que vai ser mantido todos os outros blocos de modo contínuo. Para garantir a auto disciplina, existem algumas táticas que podem ajudar.

2.5.1 Gerenciamento de Projeto

Em desenvolvimento de *software* a construção é muitas vezes complexa, então é necessário que haja um gerenciamento adequado, de forma especializada e integrada para apresentação de resultados, principalmente voltado a prazo e orçamentos.

O gerenciamento de projetos é a aplicação de habilidades, técnicas e conhecimentos voltados para a execução de projetos de forma eficaz e efetiva, utilizando uma forma estratégica para organizações, para que os resultados

adquiridos nos projetos, assim como seus objetivos, sejam utilizados para competir no mercado de trabalho. Atualmente existem diversos sites de gerenciamento de projeto como o trello, asana, jira e muitos outros. Esses sistemas possibilitam organizar as necessidades e critérios do projeto para uma melhor administração com possiblidade de datar para que o controle seja mais preciso.

A utilização de ferramentas de gerenciamento é uma boa prática, pois é possível modificar de forma maleável conforme necessário, e através de lembretes digitais, a probabilidade de perder prazos se torna menor. Porém, em alguns casos, é necessário que haja uma pessoa responsável pela administração desses sistemas de gerenciamento para controle absoluto.

2.5.2 Versionamento

Em projetos de desenvolvimento de programas é considerado boa prática fazer o versionamento de *software*, que os arquivos referentes a cada entrega devem ser encapsulados e publicados, essas entregas são as versões. Este processo ajuda a documentar inclusões, alterações e exclusões conforme necessidade do cliente ou da empresa desenvolvedora do sistema. Esse processo informa a data que cada componente foi implementado no sistema, sem falar da segurança para os utilizadores, devido a problemas de atualizações, onde restaurar a versão anterior é fácil e rápida.

A necessidade de criações de novas versões normalmente são devido a necessidade de correções de bugs, inclusão ou exclusão de funcionalidades, mudança de arquitetura ou refatoração de código e para correções pequenas e ajustes estéticos.

Para facilitar este processo, é recomendado que se utilize ferramentas para realizar o versionamento, pois é possível verificar através dela o ciclo de vida do projeto através dos históricos de alteração separados por versões assim como seus respectivos documento.

Normalmente cada desenvolvedor utiliza seu critério para realizar a semântica do versionamento, porém é recomendado que se utilize a semântica de MAJOR.MINOR.PATCH, onde a versão maior (MAJOR) é alterado quando há alterações referentes a API, a versão menor (MINOR) é modificada quando se adicionam funcionalidades mantendo a compatibilidade e a versão de correção (PATCH) quando houver correção de falhas mantendo a compatibilidade. As versões de atualização maior, menor e correção são separadas por pontos.

2.5.3 Scrum

Esta metodologia serve para gestão e planejamento de projetos de *software* onde os projetos são divididos em ciclos chamados *Sprints*, que são um time box dentro do qual um conjunto de atividades deve ser executado.

As etapas funcionam do seguinte modo. As funcionalidades a serem implementadas no projeto são armazenadas em uma lista chamada *Product Backlog*. Na iniciação de cada ciclo de projeto (*Sprint*), é feio um uma reunião de planejamento (*Sprint Planning Meeting*), onde o dono do produto prioriza os itens do que está pendente de desenvolvimento (*Product Backlog*) e a equipe seleciona as atividades para a implementação durante o ciclo. As tarefas já definidas são transferidas para o processo de desenvolvimento (*Sprint Backlog*).

A cada dia de um ciclo, é realizado uma reunião diária (*Daily Scrum*), com o intuito de realizar um momento de compartilhamento de conhecimento referente às atividades do dia anterior, voltado especificamente para reconhecer os empecilhos e priorizar o trabalho do dia atual.

No final do ciclo, a equipe expõe as funcionalidades implementadas em uma reunião de revisão (*Sprint Review Meeting*). Por fim, é realizado uma retrospectiva dos componentes implantados nesse ciclo (*Sprint Retrospective*) e a equipe inicia um novo ciclo.

2.5.4 Equipes Auto Gerenciadas

A equipe auto gerenciada é um grupo de pessoas que compartilham de forma colaborativa a responsabilidade para a realização de um objetivo. Neste modelo de equipes, todos os membros do grupo são capacitados e tem autoridade para tomada de decisões e senso crítico suficiente para se auto supervisionar.

Para este tipo de organização, a confiança mutua, exercício de liderança informar e troca de experiências e informações é uma característica muito marcante, pois todos seus membros devem ter capacidade técnica e analítica, inteligência emocional, auto comprometimento e auto controle para que haja o alcance de metas da empresa.

Os papeis dentro da equipe são distribuídos entre coordenadores e organizadores que são responsáveis por estabelecer a estrutura de equipe, definindo os demais responsáveis. O produtor, que planeja e direciona ações e alinha esforços. O controlador analisa os detalhes e garante que as regras sejam atingidas por todos. O mantenedor representa a equipe externamente. E pro fim, o conselheiro, que orienta a equipe ou busca fontes acessíveis, com a finalidade de auxiliar sob falhas, conflitos, crises e desafios.

2.5.5 Pontos Por Função

Criada em 1979, é utilizada pela indústria de *software* para medir os projetos, considerando a funcionalidade sob o ponto de vista do usuário e independente da linguagem de programação e suas tecnologias.

O principal objetivo é medir a funcionalidade solicitada pelo cliente para medir o custo com base no tamanho do projeto. Deste modo é possível reduzir os problemas relacionados ao prazo.

3 HIPÓTESE

A ideia surgiu no ambiente de trabalho, onde verifiquei um estresse sobre o código produzido. Era um código levemente confuso, levando em conta que manipulado por mais de um desenvolvedor, era complexo de mais para que um não responsável pela tarefa fizesse as correções, e ao analisar o ambiente, foi possível verificar fatores preocupantes, como:

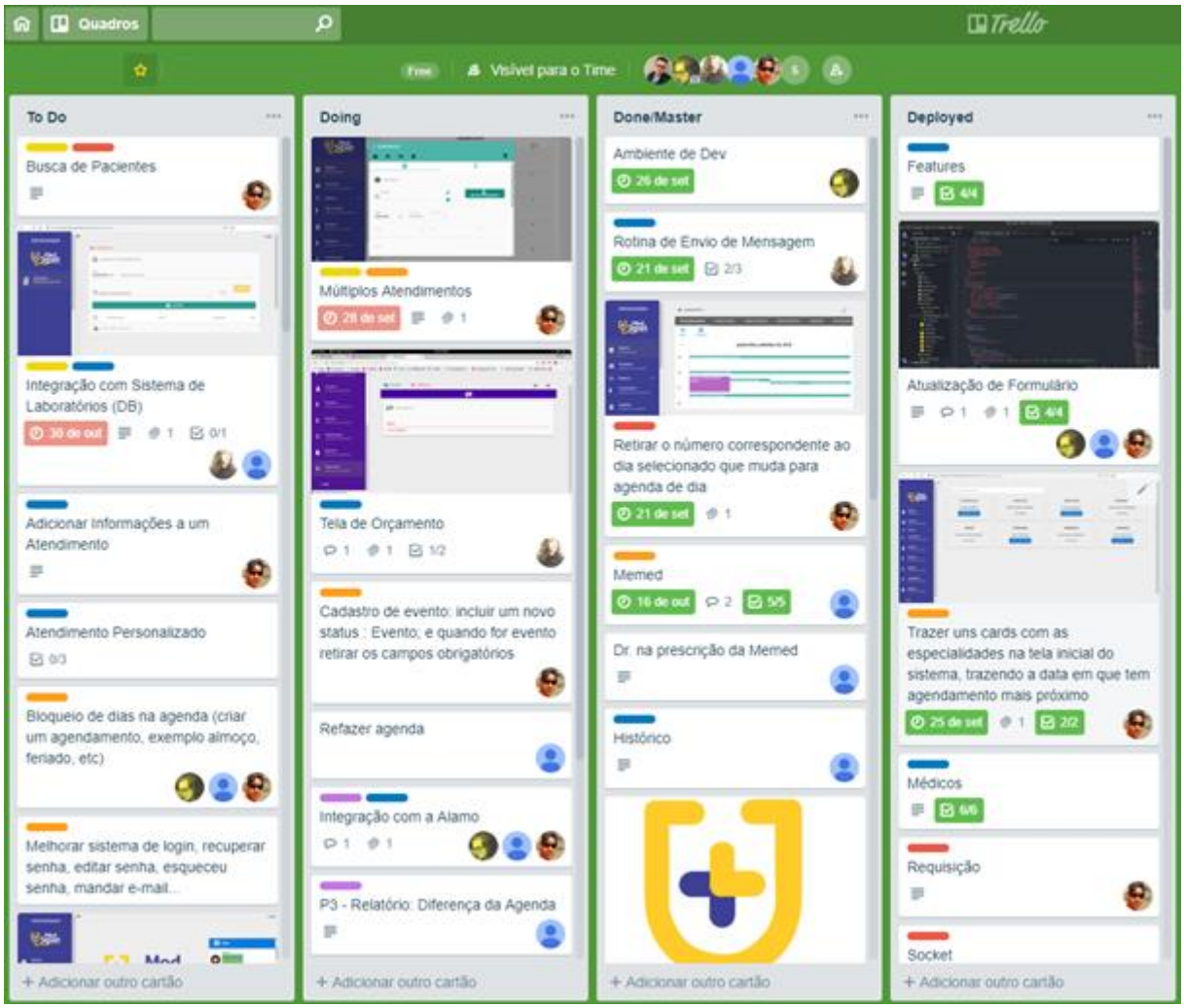
- Baixa produtividade associada ao estresse causado ao programador ao realizar ações sem pausas periódicas;
- Falta de utilização de práticas essenciais para estrutura do código;
- Mal posicionamento do programador frente ao computador;
- Falta de cronograma/gerenciamento das tarefas que estão em execução ou deveriam ser executadas;
- Erro no levantamento de requisitos, colocando módulo desnecessário no sistema;
- Problemas de calculo em pontos por função, ocasionando estresse no programador relacionado à falta de tempo para produzir.

Como o modelo 5S é empregado em diversas empresas e funciona como base de implantação para qualidade total, surgiu à ideia de que seria necessário alterar algumas atitudes para melhoria de desempenho.

Foi implantado o método Pomodoro, para gerenciamento individual de cada funcionário, garantindo as pausas periódicas necessárias para evitar os problemas ergonométricos e garantir que o funcionário consiga executar seu serviço sem estresse.

Os problemas ocasionados por mal gerenciamento foram tratados com cronogramas. Atualmente a empresa utiliza o site Trello, para gerenciamento de atividades, que é um dos mais completos e fáceis de utilizar.

Imagem 8 - Trello



Fonte: Jonathan Jonys Miranda

A melhoria de produtividade foi comprovada pelos funcionários e diretoria no decorrer de dois meses com as boas práticas sugeridas, gerando aumento de clientes para produção de *softwares* e uma satisfação dos clientes que já pertenciam a empresa.

CONSIDERAÇÕES FINAIS

A organização da empresa é um processo muito complexo que deve ser realizado com muita cautela para evitar problemas com clientes e adquirir uma má reputação neste processo.

Os 5S adaptados para programadores estão garantindo a qualidade de serviço utilizando as boas práticas, de modo que cada etapa executada corretamente garanta a qualidade dos códigos realizados.

Até o presente momento, o emprego dos 5S voltados para programadores está funcionando de forma esperada, o método se comprova eficaz na implantação, sendo uma ótima alternativa para organização dos padrões estruturais de empresas e programadores que trabalham com desenvolvimento de código.

Portanto, este novo método é completamente relevante a se empregar nas empresas, já que é necessário empregar as boas práticas para os programadores de um modo que seja seguido à risca. O 5S garante a disciplina, já que quando estamos em um ambiente que segue os padrões organizacionais, todo o contexto funciona corretamente.

A finalidade principal desta pesquisa exploratória com abordagem qualitativa, o modelo de qualidade 5S, visa a melhoria continua de codificação, propondo boas práticas para garantir a qualidade total, fortificada através de análise de documentos, pesquisas bibliográficas e aplicação em campo.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: informação e documentação: referência e elaboração. Rio de Janeiro, 2002.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6024**: informação e documentação: numeração progressiva das seções de um documento: apresentação. Rio de Janeiro, 2012.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6028**: informação e documentação: resumo: apresentação. Rio de Janeiro, 2003.

ARQUITETURA DE PROJETOS VUE.JS COM DDD. Disponível em: <https://blog.codecasts.com.br/arquitetura-de-projetos-vue-js-com-ddd-a2bc26817793> Acessom em: 06 de Novembro de 2018.

ARQUITETURA – O PRINCIPIO DA RESPONSABILIDADE ÚNICA. Disponível em: <https://www.devmedia.com.br/arquitetura-o-principio-da-responsabilidade-unica/18700> Acesso em: 06 de Novembro de 2018.

BAIXO ACOPLAMENTO. Disponível em: <https://celodemelo.wordpress.com/2009/02/10/baixo-acoplamento/> Acesso em: 20 de Novembro de 2018

DESENVOLVIMENTO BASEADO EM COMPONENTES – REVISTA JAVA MAGAZINE 10. Disponível em: <https://www.devmedia.com.br/desenvolvimento-baseado-em-componentes-revista-java-magazine-110/26550> Acesso em: 06 de Novembro de 2018.

LESÃO POR ESFORÇO REPETITIVO: SINTOMAS, TRATAMENTOS E CAUSAS. Disponível em: <https://www.minhavida.com.br/saude/temas/lesao-por-esforco-repetitivo> Acesso em 20 de Novembro de 2018.

O QUE É COMPONENTIZAÇÃO DE SOFTWARE?. Disponível em: <https://www.oficinadanet.com.br/post/14780-o-que-e-componentizacao-de-software> Acesso em: 06 de Novembro de 2018.

O QUE É GERENCIAMENTO DE PROJETOS?. Disponível em: <https://brasil.pmi.org/brazil/AboutUs/WhatIsProjectManagement.aspx> Acesso em: 07 de Novembro de 2018.

O QUE SÃO DESIGN PATTERNS?. Disponível em: <https://www.princiweb.com.br/blog/programacao/design-patterns/o-que-sao-design-patterns.html> Acesso em: 06 de Novembro de 2018.

PONTOS DE FUNÇÃO. Disponível em: <http://micreiros.com/analise-de-pontos-de-funcao/> Acesso em: 20 de Novembro de 2018.

SCRUM. Disponível em: <https://www.desenvolvimentoagil.com.br/scrum/> Acesso em: 07 de Novembro de 2018.

SOLID – SINGLE RESPONSIBILITY PRINCIPLE – SRP. Disponível em: <http://www.eduardopires.net.br/2013/05/single-responsibility-principle-srp/> Acesso em: 06 de Novembro de 2018.

VERSIONAMENTO DE SOFTWARE, SABE COMO FAZER? APRENDA AQUI!. Disponível em: <https://gaea.com.br/sabe-fazer-versionamento-de-software/> Acesso em 07 de Novembro de 2018.

VERSIONAMENTO SEMÂNTICO 2.0.0. Disponível em: <https://semver.org/lang/pt-BR/> Acesso em: 07 de Novembro de 2018.