

Fronteiras da Engenharia de Software - Podcast

Episódio 3, Temporada 3: Sistemas Altamente Configuráveis, com Márcio Ribeiro (UFAL)

Host: Adolfo Neto (UTFPR)

Co-host: Maria Claudia Emer (UTFPR)

Equipe: Danilo Ribeiro (Zup), Leonardo Fernandes (IFAL), Fabio Petrillo (ETS Montreal), Myrian Rodrigues da Silva (USES/UFAM), Italo Santos (Northern Arizona University, USA), Diego Andrade (UFC)

<https://fronteirases.github.io/>

Título: Sistemas Altamente Configuráveis, com Márcio Ribeiro (UFAL)

Resumo para Anchor e YouTube:

Márcio Ribeiro é professor Associado do Instituto de Computação da Universidade Federal de Alagoas (UFAL).

Página: <https://sites.google.com/a/ic.ufal.br/marcio/>

Lattes: <http://lattes.cnpq.br/9300936571715992> (Bolsista de Produtividade em Pesquisa do CNPq - Nível 1D)

Orcid <https://orcid.org/0000-0002-4293-4261>

Scholar: <https://scholar.google.com/citations?hl=pt-BR&user=-eYOaGwAAAAJ>

Atualmente é coordenador da Comissão Especial de Engenharia de Software (CEES) <http://comissoes.sbc.org.br/ce-es/comite.php?lang=pt-br>

Ganhou prêmios internacionais (como o ACM John Vlissides Award) e nacionais (como o Concurso de Teses de Dissertações (CTD) do Congresso Brasileiro da Sociedade Brasileira de Computação (CSBC 2013)).

<https://www.sigplan.org/Awards/Other/>

Já publicou diversos artigos em periódicos e conferências top-ranked do mundo na área de Engenharia de Software, como TSE, TOSEM, EMSE, JSS, IST, ICSE e FSE.

Em 2014, foi um dos Coordenadores Gerais do maior congresso de software do país organizado pela Sociedade Brasileira de Computação (SBC), i.e., o Congresso Brasileiro de Software (CBSOFT 2014), realizado em Maceió-AL.

Em 2024, estará na organização do ESEC/FSE.

Idealizador, criador e desenvolvedor de um aplicativo pioneiro no Brasil para cadastro, identificação e rastreamento de aglomerações de pessoas. O aplicativo foi utilizado por pessoas de todo o Brasil (exceto do estado de Rondônia) bem como por autoridades, que afirmaram que o mesmo foi útil para ajudar a polícia a identificar aglomerações e enviar contingente policial para dispersá-las, ajudando, dessa forma, no combate ao COVID-19. O projeto ganhou grande notoriedade na imprensa local, regional e nacional (TV e internet).

QUESTÕES:

FOTO



Script do Episódio

Parte 1: Apresentação [5 min, estimativa]

[ADOLFO] Olá, eu sou Adolfo Neto, professor da UTFPR Curitiba.

Hoje temos aqui, como co-host do Fronteiras, Maria Claudia Emer, também professora da UTFPR Curitiba. Tudo bem, Maria Claudia?

[MARIA] (responde e dá oi aos ouvintes)

[ADOLFO] Hoje iremos conversar com Márcio Ribeiro, professor da Universidade Federal de Alagoas. O tema do episódio de hoje será "Sistemas Altamente Configuráveis". Tudo bem, Márcio?

Você pode se apresentar para as pessoas que nos escutam?

- Márcio Ribeiro
- Professor Associado da Universidade Federal de Alagoas (UFAL), Maceió-AL
- Agradecer o convite, dizer que estou muito feliz em estar aqui com vocês, que sou ouvinte do Podcast e que é uma honra poder contribuir!
- Fiz meu Doutorado na Universidade Federal de Pernambuco (UFPE), sob a orientação dos professores Paulo Borba (UFPE) e Claus Brabrand (IT University of Copenhagen)
- Ministro disciplinas na graduação e na pós-graduação de computação, mais especificamente Estruturas de Dados e disciplinas de Engenharia de Software, como testes de software
- CBSOFT 2014
- Atual coordenador da CEES
-

Parte 2 + Parte 3 - Tema do Episódio + Pesquisa

[MARIA] Márcio, você pode começar explicando pra gente o que são sistemas altamente configuráveis?

- Sistemas altamente configuráveis são sistemas que possuem opções de configurações (conhecidas por features) para adaptar propriedades funcionais e não funcionais às necessidades de um usuário ou conjunto de usuários.
 - Em outras palavras...
 - Sistemas que possui um núcleo e também várias opções de configurações
 - E aí, podemos...
 - ... habilitar ou desabilitar essas features
 - Atendendo necessidades específicas de usuários
 - Um usuário quer somente a feature A habilitada, já um outro usuário quer várias outras features habilitadas (podendo, inclusive, pagar mais caro por isso).
 - Em outras palavras... dado um conjunto de opções de configurações (features): A, B e C, é possível gerar 8 diferentes sistemas: {}, {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}.
 - Temos 8 potenciais configurações
 - E estas podem servir a diferentes usuários, clientes etc
- Exemplos:
 - Nível de funcionalidades: Office

- Nível de hardware: Impressoras
- Nível de plataforma e sistema operacional: 32 bits versus 64 bits
 - Tamanho da palavra: 4 bytes versus 8 bytes
- Possuem suporte ferramental para lidar com as features...
 - Modelos que indicam quais features são obrigatórias (editor de texto), opcionais (corretor em determinada língua), alternativas (sistema do posto só funciona com uma impressora por vez; escolhendo a impressora A implica que não se pode escolher a impressora B etc)
 - Ferramentas para a geração e compilação de sistemas dado um conjunto de features etc
- Importante salientar que esses sistemas não são novos... são muito antigos, na verdade
 - Foram mais explorados pela academia um pouco recentemente

[ADOLFO] E como são implementados estes sistemas?

- Existem muitas técnicas
- Técnicas de OO (herança e polimorfismo de subtipo e paramétrico), Padrões de projetos, arquivos de configuração, leitura de bibliotecas de forma dinâmica, pré-processadores etc
- Técnica bastante utilizada: pré-processadores
- O que são esses pré-processadores?
 - Marcação do código com anotações
 - int x = 0; para o Windows
 - int x = 1; para o Linux
 - #ifdef WIN int x = 0;
 - #ifdef LINUX int x = 1;
 - Antes de compilar, há um pré-processamento
 - Se for WIN, o código será compilado com x = 0; se for LINUX, o código será compilado com x = 1.
- Como vocês podem notar, trata-se de uma técnica super simples de usar. Fácil e rápido! É um comando parecido com um if, só que em vez de ser exercitado em tempo de execução, é em tempo de pré-compilação
- Utilizada em diversos sistemas, como sistemas operacionais (a exemplo do kernel do Linux), servidores web como o Apache, editores de texto (como o vim), SGBDs como o postgresql etc.

[MARIA] Como você falou acima, os #ifdefs representam uma técnica muito simples para implementar variabilidade em sistemas configuráveis. Mas há desvantagens? Se sim, poderia comentar um pouco sobre isso?

- Usos isolados, OK
- Vários #ifdefs, usos aninhados, pode-se ter problemas de legibilidade e entendimento
- Falar do exemplo do vim abaixo:

```

#if defined(FEAT_XCLIPBOARD) || defined(USE_XSMP) || defined(FEAT_MZSCHEME)
    static int busy = FALSE;

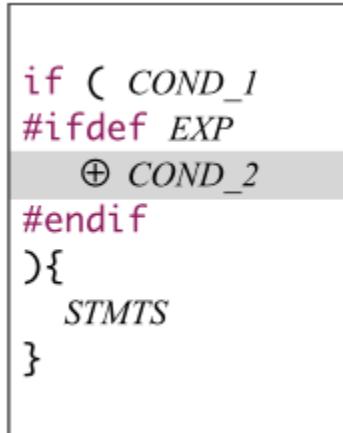
#if defined(HAVE_GETTIMEofday) && defined(HAVE_SYS_TIME_H)
    if (msec > 0 && (
#endif FEAT_XCLIPBOARD
    xterm_Shell != (Widget)0
#if defined(USE_XSMP) || defined(FEAT_MZSCHEME)
    ||
#endif
#endif
#endif
#endif USE_XSMP
    xsmp_icefd != -1
#endif FEAT_MZSCHEME
    ||
#endif
#endif
#endif FEAT_MZSCHEME
    (mzthreads_allowed() && p_mzq > 0)
#endif
))
    gettimeofday(&start_tv, NULL);
#endif
    if (busy)
        return 0;
#endif

```



"Vim is a highly configurable text editor built to make creating and changing any kind of text very efficient. It is included as "vi" with most UNIX systems and with Apple OS X."

- Manutenção bem difícil para este caso
- Para piorar: pessoas usam de forma não disciplinada
- Explicar o que é isso em um exemplo



- Em suma:
 - Heaven:
 - Simples

- Sem overhead
- Resolve problemas de portabilidade
- Hell:
 - Susceptibilidade a erros
 - Problemas de entendimento e legibilidade
 - Problemas de manutenção



[ADOLFO] A indústria e o mundo open source estão cientes do que são anotações não disciplinadas?

- Open source:
 - Falar que com outra terminologia, aparentemente sim
 - Falar do manual do Linux:

"Prefer to **compile out entire functions, rather than portions of functions or portions of expressions."**

- Outro exemplo é do libpng: "I agree. In fact, it's in the libpng17 TODO list: Refactor preprocessor conditionals to compile entire statements."
- Academia:
 - Artigo do AOSD 2011, Jorg Liebig, Christian Kästner e Sven Apel
 - Anotações disciplinadas e não disciplinadas
 - Vários exemplos
 - Estudo de 40 sistemas configuráveis
 - 84% disciplinadas e 16% não disciplinadas
 - Mas, estudos em relação a esses tipos de anotações parecem ser mais antigos: Artigo do ASE 2003, Alejandra Garrido e Ralph Johnson...
 - Anotações completas e incompletas
 - A definição parece que possui a mesma essência
 - Iniciamos um estudo em relação aos 10 anos, resultados preliminares...
 - Dia (desenho de diagramas), houve um aumento de quase 12% no número de anotações disciplinadas
 - Libssh aumento de 5%
 - Fazendo uma média geral... aumento para 90%

- Pergunta é: esse aumento foi uma contribuição da academia?

[MARIA] Como é feita a manutenção de sistemas configuráveis?

- Certamente a manutenção se torna mais complicada, pois agora você não está mais dando manutenção em apenas um sistema para apenas um cliente; mas sim para um conjunto (que pode ser enorme) de sistemas para vários clientes diferentes com suas especificidades e peculiaridades
- Outro problema é que é mais difícil encontrar bugs: um problema (sintático ou semântico) só poderá ser identificado caso a configuração exata em que o problema existe seja exercitada (compilada, por exemplo)
- A manutenção é mais difícil... (2^n - combinações de features proibidas)
 - $2^{33} = 8$ bilhões
 - $2^{(mais de 12.000)} = ?$
- Em um sistema grande, inviável...
- Mas claro que para muitos sistemas, apenas um conjunto interessa (clientes)
- Um dos estudos que fizemos: caso do kernel, onde o desenvolvedor falou:

"I check whatever combinations I can, and some combinations can only be tested on systems to which I have no access. I rely on others to help out or just cross my fingers."

- Caso do libssh, onde achamos que tínhamos encontrado um problema, mas não era!

```

1. #ifdef HAVE_LIBCRYPTO
2. static void dsa_public_to_string(gcry_sexp_t key, BUFFER *buffer) {
3. #elif defined (HAVE_LIBCRYPTO)
4. static void dsa_public_to_string(DSA *key, BUFFER *buffer) {
5. #endif
6.   // code
7. }
```



- Outro problema: os testes precisam ter configurações também

[ADOLFO] Parece que as anotações não disciplinadas que você comentou têm relação com code smells. O que você acha sobre isso? Já foram sugeridas refatorações para a remoção desses smells?

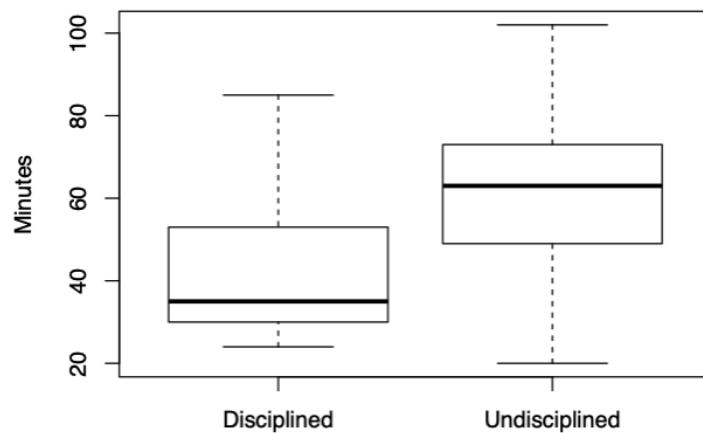
- Sim, de acordo

- São pedaços de código que não estão com um bom design e que, potencialmente, podem levar a problemas futuros, como introdução de bugs
- As anotações não disciplinadas parecem se encaixar bem nessa definição. Elas não estão cheirando bem e, devido a problemas de legibilidade e entendimento que elas podem trazer, os desenvolvedores podem introduzir bugs
- Para tentar minimizar esses problemas, tem-se as refatorações.
- Já foram propostos algumas. Dado um lado esquerdo não disciplinado, a refatoração gera um lado direito com uma anotação disciplinada
 - Return
 - Chamadas de funções
 - While, if...
 - Arrays, enums

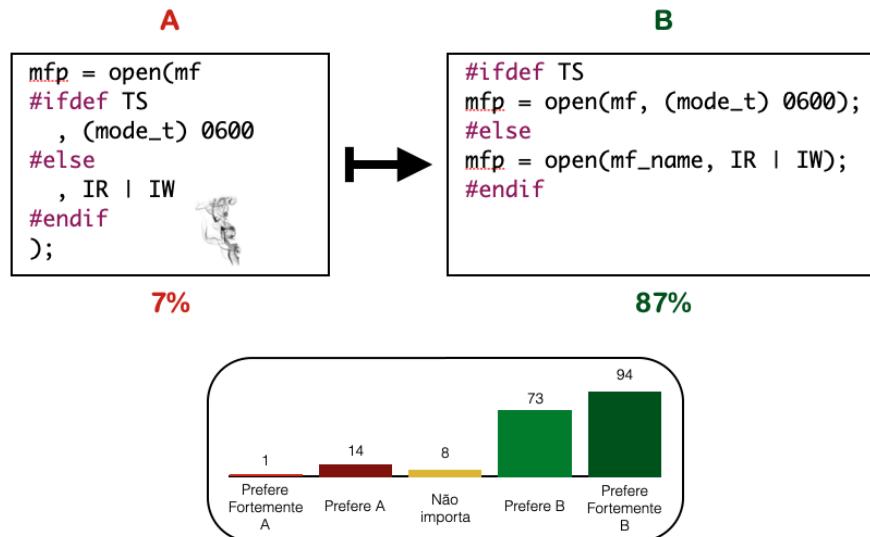
CORTE 34:50

[MARIA] Mas como avaliar essas refatorações? Como saber se elas realmente trouxeram ou podem trazer ganhos?

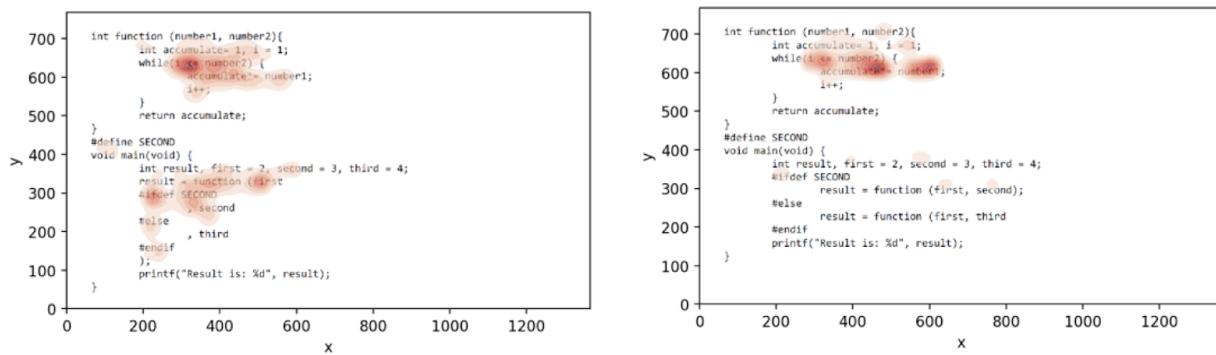
- Estudos controlados diversos
- Estudos controlados com medição de tempo e número de erros cometidos pelos participantes



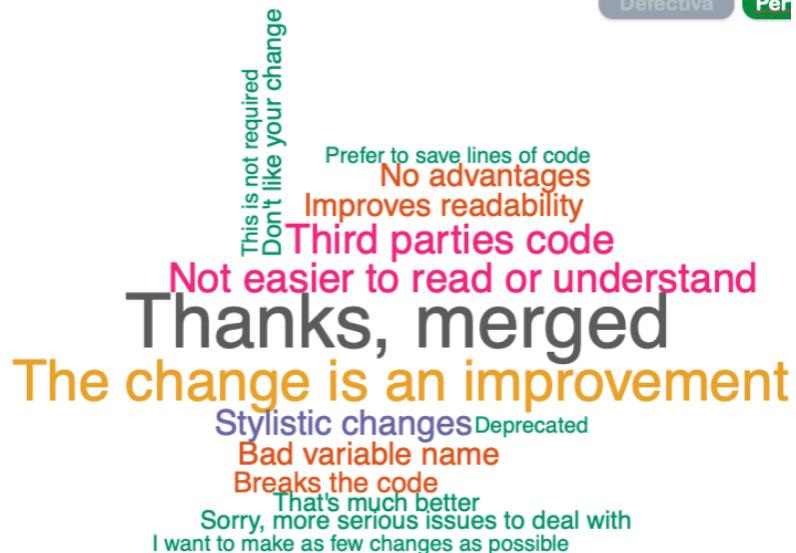
- Survey com desenvolvedores
 - Lado esquerdo
 - Lado direito
 - A gente sempre fez de forma aleatória os lados



- Eye tracking
 - Mapa de calor
 - AOI



- Contribuições em sistemas open-source: Pull requests



(EXCLUÍDOS PARA NÃO FICAR MUITO LONGO)

[ADOLFO] Quais são os pontos positivos e negativos em trabalhar com Sistemas Altamente Configuráveis, o que você sugere que precisa ser mais investigado na área?

Certamente, um ponto positivo é a grande variedade de produtos que você pode oferecer. Com isso, você pode atingir vários clientes, usuários etc. Além disso, seu produto de software pode atingir diversos ambientes (sistemas operacionais, por exemplo).

Acho que seria importante investigar e trazer mais suporte ferramental. Existe um conjunto de ferramentas juntas em um só ambiente de desenvolvimento, que é a FeatureIDE, inclusive com contribuições nossas da UFAL. Seria importante tentar difundir mais essa e outras ferramentas, criar e avaliar novas.

[MARIA] Que dificuldades você enfrenta ao conduzir estudos experimentais em suas pesquisas e como você resolve?

Primeiramente, eu diria que é recrutar as pessoas corretas para o estudo. Mas um outro grande desafio é escolher as tarefas corretas de forma que elas, de fato, avaliem as hipóteses do seu estudo.

Isso me lembra um keynote do Professor Fernando Castor da UFPE no workshop VEM do CBSoft. Naquele workshop, Castor mencionou importantes pontos nesse sentido no contexto de entendimento de programas. Qual é a melhor tarefa para, de fato, avaliar seu estudo?

- Dizer a saída de um determinado código?
- Adicionar uma linha faltando em um código ou algoritmo qualquer?
- Dizer o valor quer será impresso de uma determinada variável?

Parece-me que definir bem isso é um grande desafio!

[ADOLFO] Você e coautores tiveram um artigo publicado na IEEE Transactions on Software Engineering, em 2022, chamado "Refatorando Maus Cheiros de Teste com JUnit 5: Por que os desenvolvedores devem se manter atualizados". ([\("Refactoring Test Smells With JUnit 5: Why Should Developers Keep Up-to-Date"\)](#)). Você pode falar um pouco sobre este artigo?

- Primeiramente, falar de test smell

```

1  @Test
2  void first_test() {
3      if (lastContainerId == null) {
4          lastContainerId = genericContainer.getContainerId();
5      } else {
6          assertEquals(lastContainerId,
7              genericContainer.getContainerId());
8      }
9 }
```

Listing 1: The Conditional Test Logic smell

- Notamos que vários pesquisadores têm tentado remover smells de teste utilizando refatorações já propostas. Por exemplo, para remover código de teste duplicado, aplicar um extract method
- Mas, notamos também que frameworks como o JUnit 5 possuem um conjunto de anotações em que é super rápido e prático remover os smells
 - `@ParametrizedTest`
 - `@EnableIf`
 - `@RepeatedTest`
- Criamos refatorações para remover esses test smells usando anotações do JUnit 5
- Avaliamos com um survey, onde a maioria esmagadora escolheu o lado com as nossas refatorações
- Submissão de contribuições com pull requests, onde a maioria também aceitou as nossas propostas

[MARIA] Você teve um artigo aceito, em colaboração com outras pessoas como o nosso colega de Fronteiras Leo Fernandes, no (Simpósio Brasileiro de Engenharia de Software SBES 2022):

"Ponha suas mãos no ar! Reduzindo o esforço manual nos testes de mutação".
Você pode falar um pouco para nós sobre este artigo?

- Artigo que está no contexto de testes de mutação
- Identificar mutantes equivalentes, que, na verdade, é um problema indecidível
- Tarefa manual: dado dois mutantes, dizer se eles são equivalentes ou não
- Suscetível a erros e demanda muito tempo
- Geração automática de testes... geramos um monte de testes e executamos no programa original e no mutante
- Suite de teste é verde, passa no código original. Assim, se um teste matar o mutante, quer dizer que o mutante não é equivalente
- Para a avaliação, usamos um conjunto de mutante manualmente anotados como equivalentes ou não
- Nossa abordagem teve uma acurácia alta, 96%, ou seja, o que ela disse que era equivalente ou não equivalente, realmente era.
- Como é uma abordagem automática, ela levou $\frac{1}{3}$ do tempo para indicar equivalência e foi 25 vezes mais rápida para indicar não equivalência

Parte 4: Outras perguntas

[ADOLFO] Você é atualmente coordenador da Comissão Especial de Engenharia de Software (CEES) da Sociedade Brasileira de Computação (SBC). Quais são os desafios atuais como coordenador da CEES?

- Falta de documentação dos processos da CEES, tudo em e-mails e nas memórias das pessoas
- Criamos:
 - Regimento da CEES
 - Alguns manuais de organização de eventos (CBSOFT, SBES etc)
 - Drive para agrupar todas as informações
- Indexação do SBES 2006, 2007 e 2008 (faltam 22 anos!)
- Dados dos eventos de ES para a quadrienal da CAPES
- Carta para ajudar a trazer o ICSE 2026 para o Brasil
- Organização presencial do CBSOFT 2023
- Documento para o patrocínio da ACM SIGSOFT (Escola LATAM)

Parte 5: Próxima Fronteira da ES [3 min, estimativa]

[ADOLFO] Para você, qual é a próxima fronteira da engenharia de software? (pode ser algo que você acha que vai acontecer ou que você gostaria que acontecesse em nossa área)

- Dar o melhor suporte possível às aplicações contemporâneas, que estão cada vez mais complexas
- Indústria 4.0
- Inteligência Artificial, modelo, flaky...
- IoT

Parte 6: Encerramento

[ADOLFO] Agradece e passa para o(a) entrevistado(a).

[MARIA] Fecha o episódio.