

Aumentando a produtividade!

Produtividade no desenvolvimento
de sistemas empresariais utilizando
tecnologias atuais

Sobre o Palestrante

- Rodrigo Urubatan - SCJP 1.4 e SCWCD
- Trabalha com arquitetura de sistemas J2EE e treinamento
- Já desenvolveu projetos utilizando as linguagens Delphi, C++, PHP, ASP, ColdFusion, Leather, Assembly, Perl, ...
- Trabalha com Java/J2EE a 4 anos e com desenvolvimento de sistemas a 9 anos
- Atualmente colabora com pequenas correções a alguns projetos Open Source como o GUI2, Lombos e Veloclipse e faz parte da coordenação do RSJUG
- Já ministrou palestras em universidades (UCS, ULBRA, UNISC) e diversos eventos (Just Java, FISL, Seminário do RSJUG, Maratona 4 Java, Infosul) tutoriais para o RSJUG e já teve um artigo publicado na revista Mundo Java
- Atualmente trabalha como consultor na AdvancedIT, como gerente de tecnologia e qualidade na Tech Office IT, e ministra cursos e alguns pequenos projetos pela USI Informática
- É o principal desenvolvedor do projeto Spring-Annotation

O que é produtividade

- Entregar o sistema no prazo para o cliente
- Entregar o sistema sem BUGs
- Facilitar a manutenção do sistema
- Atingir as metas com menos esforço

Impecilios para a produtividade

- Trabalho demais para atingir um objetivo simples
- Trabalho repetitivo
- Utilização de tecnologias defasadas
- Falta de comunicação
- Demora na detecção de problemas

Problemas da falta de produtividade

Cliente Insatisfeito!

Como melhorar a produtividade no desenvolvimento de softwares?

- Utilizando tecnologias atuais
 - Um dos principais motivos para a evolução das tecnologias de desenvolvimento de sistemas é aumentar a produtividade dos desenvolvedores
- Diminuindo os ciclos de desenvolvimento e apresentando o progresso do desenvolvimento aos clientes
- Utilizando novas metodologias de desenvolvimento

Problemas da utilização de tecnologias ultrapassadas

- Mais trabalho do que o necessário
- Dificuldade para testar o que foi desenvolvido
- Pouca ajuda das novas ferramentas

Problemas encontrados

- Menos experiência dos profissionais
- Custo de aquisição de novas ferramentas
- Treinamento
- Software legado/já existente
- Resistência ao novo!

Vantagens obtidas imediatamente

- Melhora na motivação dos profissionais
- Fator “marketing” para software houses e consultorias
- Melhora na qualidade dos softwares e consequente aumento da satisfação dos clientes

Comparações

- Java EE
 - Versão 1.4
 - Para criar um EJB Stateless era necessária a criação de no mínimo 3 interfaces e 1 classe, com bastante código duplicado além de atualizar o XML do deployment descriptor
 - Para criar um MDB era necessário 1 classe 1 interface além da alteração do XML do deployment
 - Para acessar um EJB eram necessárias pelo menos 5 linhas de código e um type cast

Comparações

- Java EE
 - Versão 1.5
 - Para criar um EJB Stateless é necessário apenas 1 classe
 - Para criar um MDB é necessário apenas a classe do MDB e 1 anotação
 - Para acessar um EJB é necessário apenas a declaração de um campo do tipo do EJB anotado com @EJB

Comparações

- **Hibernate**
 - **Versão 2.0**
 - Para persistir uma classe era necessário criar a classe persistente + 1 arquivo de mapeamento XML com todos os campos da classe declarados novamente
 - **Versão 3.0**
 - Para persistir uma classe é necessário apenas a classe a ser persistida com 2 anotações

Comparações

- Spring Framework
 - Versão 1.x
 - Para acessar um data source via JNDI eram necessárias no mínimo 5 linhas em um arquivo XML
 - Versão 2.x
 - Para acessar um data source via JNDI é necessário apenas 1 linha de configuração

Comparações

- Interatividade na WEB
 - Anterior ao AJAX
 - Para cada item interativo era necessário criar uma página secundária em um iframe + todo o javascript necessário
 - Ajax
 - Utilizando alguma biblioteca JavaScript 1 linha de JavaScript
 - Utilizando em conjunto com JSF uma tag

Comparações

- Delphi
 - Versão 2
 - Para criar uma barra de ferramentas era necessário alinhar um painel, configurar o tamanho, escolher os botões, adicionar as imagens, ...
 - Acesso a dados apenas via BDE, mais ma instalação para a maquina do cliente
 - Versão 7
 - Componente pronto
 - Componentes de acesso nativo a diversos bancos de dados

Ciclo de Vida de um software

- Na maioria dos casos fica em torno de
 - 30% desenvolvimento
 - 70% manutenção

Soluções

- A maior parte das novas tecnologias tem parte do foco na escrita de menos código para realizar o mesmo trabalho
- Menos código escrito quer dizer menos possíveis pontos de falha
- Menos possíveis pontos de falha quer dizer menos pontos para dar manutenção

Sugestões

- Verificar a possibilidade de atualizar a tecnologia
- Na impossibilidade, analisar os ganhos da migração para novas tecnologias
- Utilizar tecnologias atuais nos novos projetos
- Verificar a compatibilidade para utilizar tecnologias atuais nas manutenções de sistemas antigos

Muito Obrigado!

Urubatan