O Progamador Pragmático:

De Aprendiz a Mestre

•••

Capítulo 8: Projetos Pragmáticos

PET/ADS

INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos

Este material foi desenvolvido pelo grupo PET/ADS do IFSP São Carlos

Introdução

À medida que o projeto avança, é necessário deixar de lado questões de filosofia e codificação individuais para focar em questões mais amplas relacionadas ao tamanho do projeto, a fim de abordar áreas críticas que podem contribuir ou prejudicar o projeto.

Equipes Pragmáticas

Há vantagens em ser um indivíduo pragmático, mas essas vantagens aumentam muito quando se está trabalhando em uma equipe pragmática.

Vamos reformular as seções anteriores, agora abrangendo equipes.

Sem janelas quebradas

> “A qualidade só pode advir das contribuições individuais de todos os membros da equipe”.

Cada indivíduo da equipe deve ser encorajado a conhecer e implementar a filosofia da erradicação de janelas quebradas, para evitar o óbice da deterioração do código. A explicação dessa filosofia se encontra no primeiro capítulo.

Sapos cozidos

Certifique-se de que todos os membros da equipe estão monitorando ativamente alterações no ambiente (mudanças no escopo, diminuição nas escalas de tempo, recursos adicionais ou qualquer coisa que não faça parte do que foi originalmente acordado.)

Não é necessário rejeitar alterações concluídas, apenas precisa notar que elas estão ocorrendo, caso contrário a equipe será cozida sem nem ao menos notar tal qual o sapo mencionado em Sopa de Pedras e Sapos Cozidos.

Um supervisor pode ser designado para procurar constantemente alterações no ambiente.

Comunique-se

“Para as pessoas as piores equipes são aquelas que parecem caladas e reticentes”.

Equipes de projetos profissionais devem se comunicar como uma entidade, a documentação que produzem deve ser clara, precisa e consistente, apresentam-se bem.

Consequentemente, as pessoas querem se reunir com elas.

Todos os membros da equipe devem falar a mesma língua.

Crie uma marca

Criar uma marca é um truque de marketing que apesar de parecer tolo, contribui para que a equipe fale a mesma língua.

Ao iniciar um projeto, dê um nome a equipe – preferencialmente algo incomum - crie um logo, use-o em memorandos e relatórios.

Essa simples ação dará a equipe uma identidade definida e ao resto do mundo algo memorável para associar a seu trabalho.

Não se repita

Duplicação leva ao desperdício do trabalho e pode resultar em uma manutenção complicada, uma boa comunicação pode ajudar a evitá-la, mas nem sempre é suficiente.

É possível designar um bibliotecário do projeto, esse membro ficaria responsável por coordenar os depósitos de documentos e códigos.

Um bom bibliotecário seria capaz de identificar duplicação ao manipular o material.

Em projetos muito grandes ou em que nenhum membro da equipe quer desempenhar esse papel, deve ser designado pessoas encarregadas por vários aspectos funcionais do trabalho, sistemas groupware e grupos de notícias também são muito úteis para evitar repetições.

Ortogonalidade

É um erro pensar que as atividades de um projeto - análise, design, codificação e teste - devem ocorrer isoladamente.

Esses são atributos distintos de uma mesma problemática e separá-los em blocos de responsabilidade rigorosos acarreta muitos problemas.

A principal problemática é que programadores em níveis mais distantes dos usuários provavelmente não conhecerão o contexto em que seu trabalho será usado e consequentemente não conseguirão tomar decisões embasadas

Dicas

DICA 60: Organize as equipes com base na funcionalidade.

Essa organização tende a resultar em um conjunto de desenvolvedores mais comprometidos, reduzindo escalas de tempo e aumentando a qualidade.

A organização de equipes como códigos protege-as dos efeitos de mudanças, se o usuário decidir alterar algum aspecto, apenas a equipe responsável por aquela funcionalidade será afetada.

Automação

Para garantir que tudo seja automatizado, escolha um ou mais membros da equipe para construir e instalar as ferramentas que automatizam a parte chata do projeto.

Saiba quando parar de adicionar retoques

Cada indivíduo da equipe deve brilhar de seu modo, apenas assegure que o projeto seja distribuído de acordo com seus requisitos.

Como o pintor de Software satisfatório resista à tentação de adicionar mais retoques, ou o tópico retratado se perderá na pintura.

Automação Onipresente

“A civilização avança ao aumentar o número de operações importantes que podemos executar sem pensar.”

Alfred North Whitehead.

As instruções para ligar um Ford Modelo T ocupavam mais de duas páginas, permitindo a existência de erros caso alguma etapa não fosse seguida corretamente. Nos dias atuais, o procedimento de partida é simplificado ao virar a chave, à prova de acidentes e muito mais ágil.

Para qualquer tarefa recorrente no desenvolvimento de projetos, o ideal é que seja tão simples quanto "virar a chave".

Tudo no automático

Observe o exemplo:

Um administrador de sistemas dava a cada desenvolvedor um conjunto de instruções que ocupavam muitas páginas referente a instalação de pacotes complementares em uma IDE para o desenvolvimento de um projeto. Entretanto, diferenças no comportamento do aplicativo ocorriam quando desenvolvedores distintos executavam o mesmo código. Erros apareciam em uma máquina, mas não em outras.

O processo manual pode causar discrepâncias, uma vez que permite interpretações variadas das instruções e leva a problemas.

Dicas

DICA 61: Não use procedimentos manuais

Tudo no automático: Ferramentas favoritas

- Script de Shell ou arquivos em lote: Permite executar as mesmas instruções nas mesmas ordens. Solução para o problema anterior.

- Cron (ou at no windows NT): Permite agendar tarefas automáticas para serem executadas periodicamente, como backups, construção noturna, manutenção de sites web e etc.

```
# MIN HOUR DAY MONTH DAYOFWEEK COMMAND
# ----------------------------------------------------------------
  5    0    *    *    *    /projects/Manhattan/bin/nightly
 15    3    *    *  1-5     /usr/local/bin/backup
  0    0    1    *    *    /home/accounting/expense_reports
```

O exemplo de Cron ao lado, especifica que o comando nightly de um projeto seja executado à meia noite e cinco todo dia, que o backup seja executado às 3:15 da madrugada nos fins de semana e que expense_reports seja executado à meia-noite no primeiro dia do mês.

Compilando o projeto

É recomendável compilar projetos usando makefiles, pois trata-se de um procedimento automatizado por meio de scripts, e permite a inclusão de ganchos que geram códigos e testes de regressão de forma automática.

Isso possibilita o acesso ao código, compilação, teste e distribuição com um único comando.

Gerando código

A geração de código deve obter informações nas mesmas fontes, é possível tornar esse processo mais fácil utilizando dependências de make, a partir da adição de regras a um makefile ocorre a geração de um arquivo a partir de alguma outra fonte automaticamente

```
.SUFFIXES:    .java    .class    .xml
.xml . java:
        perl convert.pl    $< >    $@
.java.class:
      $(JAVAC)    $(JAVAC_FLAGS) $<
```

Ao digitar o comando make “test.class”, o make do exemplo irá tentar encontrar um arquivo “test.xml”, convertê-lo em test.java usando o script Perl, e então compilar “test.java” para gerar “test.class”.

Testes de regressão

O makefile também faz testes de regressão para você.

Isso possibilita que você teste desde um projeto inteiro, com apenas um comando no topo da árvore de fontes, até um módulo individual usando o mesmo comando em um único diretório.

Comando make recursivo: O comando make gerencia somente as dependências que estão dentro do escopo da chamada atual, isso significa que se você não for cuidadoso suficiente, é de alta probabilidade que haja recompilações desnecessárias e perda de dependências e não recompilação quando necessário.

Automação da construção

A construção trata-se do procedimento de pegar um diretório vazio e um ambiente de compilação conhecido e construir o projeto do zero.

O ponto importante é a fazer a construção completa executar todos os testes disponíveis.

Na maioria dos projetos, o nível de construção é executado automaticamente todas as noites, ao modo que os testes são executados regularmente, se um teste de regressão resultar em algum erro, o problema será identificado no mesmo dia.

"Identificando o problema perto da origem, terá uma chance melhor de encontrá-lo e corrigi-lo."

Automação da Construção: Separação em etapas

As etapas desse processo normalmente são:

1. Extração do código-fonte do repositório
2. Construção do projeto a partir do zero: normalmente é feita a partir de um makefile superior, é marcada por algum número de lançamento, versão ou carimbo de data.
3. Criação de uma imagem de distribuição
4. Execução de testes específicos.

Construções finais

As construções finais representaram a versão final que você deseja entregar como produto, também podem possuir requisitos distintos da noturna regular.

Por exemplo, podem exigir que repositório seja marcado com o número da versão. Nestes casos, é comum empregar um objetivo separado para make (como make final) para configurar todos esses parâmetros simultaneamente.

Administrivia automática

As tarefas de um programador não se resumem a programar.

Para os trabalhos mais tediosos há a possibilidade de utilizar scripts automatizados, que processem tarefas com base no conteúdo do código fonte e da documentação.

Desse modo, você não precisaria nem se quer lembrar de executar o comando quando necessário.

Geração de sites

É comum utilizar um site interno para a comunicação no projeto, porém fazer isso de modo manual pode resultar em obsolescência ou desatualizações prejudiciais.

A solução para essa problemática é automatizar a geração do conteúdo do site a partir do repositório do projeto, podendo ser feita integrando-o à construção noturna ou ao retorno do código-fonte.

Assim, as informações do site estarão em sincronia com o código-fonte e a documentação do projeto, segundo um dos princípios de NSR, na qual as informações existem em um formato como documentos e códigos.

Procedimentos de aprovação

Utilize a automação para reduzir a carga burocrática dos procedimentos de aprovação.
É possível inserir um marcador específico em cada arquivo de código-fonte, como:

```
/* Status: needs_review */
```

Um script simples percorrerá o código fonte na busca de “needs_review” e criará uma lista web com os códigos para revisão, permitirá enviar e-mails automáticos ou agendar reuniões e até atualizar o status após a revisão.

Cuidado, em um artigo da CACM de abril de 1999, Robert Glass resume uma pesquisa que parece indicar que, embora inspecionar o código seja eficaz, conduzir revisões em reuniões não é.

Casa de ferreiro

“Casa de ferreiro, espeto de pau.”

Muitos programadores utilizam ferramentas inadequadas.
Delegue as tarefas repetitivas ao computador, há ferramentas ótimas como:

- Cron;
- Make;
- Ant;
- CruiseControl;
- Ruby;
- a Perl.

Testando Incansavelmente

Programadores pragmáticos são instigados a encontrar os erros agora, evitando a vergonha de outros encontrarem posteriormente.

Encontrar erros é como pescar com uma rede.

Utilize redes pequenas e finas (teste de unidade) para pegar peixes de água doce e redes grandes e grossas (testes de integração) para pegar tubarões.

As vezes um peixe escapa e precisamos reparar qualquer buraco encontrado na rede para evitar que outros fujam.

Peixes e Tubarões

Cuidado, pequenos peixes têm o péssimo hábito de se tornarem grandes tubarões.

Tubarões são mais difíceis de serem caçados.

Para evitar isso, comece a testar assim que tiver algum código, comece com peixes pequenos.

Dicas

DICA 62: Teste cedo. Teste sempre. Teste automaticamente.

Codifique um pouco, teste um pouco: esse deve ser o nosso mantra ao codificar, bons projetos podem ter mais código de teste do que código de produção.

Dicas

DICA 63: A codificação só estará concluída após todos os testes serem executados

O código nunca termina realmente, é um erro alegar que ele possa ser usado por alguém antes que tenha passado por todos os testes.

O que testar

Tipos importantes de teste que precisam ser executados:

- Teste de unidade
- Teste de integração
- Validação e verificação
- Exaustão de recursos, erros e recuperação
- Teste de desempenho
- Teste de usabilidade

Teste de unidade

Todos os módulos que você estiver usando devem passar pelos testes de unidade antes de avançar, se as unidades não funcionarem bem separadas muito dificilmente funcionaram juntas.

Teste de integração

Testa os principais subsistemas que compõem o projeto funcionam e interagem bem uns com os outros.

Validação e verificação

Assim que você estiver com um protótipo ou uma interface de usuário executável é essencial confirmar se o resultado atende aos requisitos funcionais do sistema.

É imprescindível garantir que o sistema esteja solucionando o problema correto, se questione, os usuários lhe disseram o que queriam, mas "é disso que eles precisam?”.

Exaustão de recursos, erros e recuperação

Você deve considerar a exaustão de recursos como:

- Memória;
- Espaço em disco;
- Paleta de cores;
- Capacidade de recuperação diante dessas limitações.

Quando o sistema falhar é importante que ele tenha um encerramento adequado, sem perda de informação.

Teste de Desempenho

Refere-se aos testes de requisitos de desempenho considerando as condições do mundo real, ou seja, com a quantidade de usuários ou conexões ou transações por segundo esperadas.

Por vezes será necessário um hardware ou software de teste especializado para simular a carga de forma realista.

Teste de Usabilidade

“Não conseguir atender os critérios de usabilidade é um erro tão grande quanto a divisão por zero.”

Esse teste é executado com os usuários reais, sob condições ambientais reais, aqui será avaliado se o projeto atende as necessidades e expectativas do usuário.

É crucial executar esse teste o mais cedo possível, aproveitando o tempo disponível para corrigir discrepâncias identificadas durante a análise de requisitos.

Como testar?

Isso inclui:

- Teste de projeto;
- Testes de regressão;
- Dados de teste;
- Testar sistemas de GUI;
- Testar os próprios testes;
- Testes completos.

Teste de Projeto

Para analisar a metodologia e o projeto de código usado, utilize métricas como:

- Complexidade ciclomática de McCabe;
- O fan-in (número de classes base);
- O fan-out (número de módulos derivados que usam esse módulo como pai) da herança;
- Conjunto de resposta;
- Taxas de vinculação de classes.

Teste de regressão

Esse teste, por meio da comparação da saída do teste atual com os valores conhecidos, verifica se as correções recentes não afetaram funcionalidades anteriores.

Pode ser usados para verificar contrato, desempenho, validade e etc.

Dados de Teste

Há dois tipos de dados de teste

- Dados do mundo real;
- Dados Sintéticos.

Dados do mundo real

São os coletados de alguma fonte real, como de algum sistema, de um protótipo, ou no sistema de um concorrente.

Normalmente eles revelam defeitos e divergências nas análises de requisitos.

Dados Sintéticos

São gerados automaticamente, podem sofrer com restrições estatísticas.

Utiliza-se dados sintéticos quando:

- Necessita-se de muitos dados;
- Precisar de dados para forçar condições limítrofes;
- Exibir certas condições estatísticas;
- Testar a face de falhas sistemáticas.

Testando sistemas de GUI (Interface Gráfica do Usuário)

Os testes de sistema que fazem uso sistêmicos de GUI com frequência requerem ferramentas de testes especializadas, como modelo de captura/reproduções de eventos ou podem requerer scripts escritos especialmente para condução da GUI, alguns subsistemas combinam os dois.

As ferramentas modernas se adaptam a pequenas diferenças de layout para evitar falhas nos testes devido a alterações visuais, como modificar o tamanho de um botão.

Contudo, nem tudo pode ser automatizado.

O caso de Andy

“Andy trabalhou em um sistema gráfico que permitia que o usuário criasse e exibisse efeitos visuais não determinísticos que simulavam vários fenômenos da natureza. Infelizmente, durante o teste não era possível apenas capturar um bitmap e comparar a saída com uma execução anterior, porque ela foi projetada para ser sempre diferente.”

Em casos como esse, se deve confiar na interpretação manual dos casos de teste.

Testando os próprios testes

Do mesmo modo que não é possível criar um software perfeito, também não podemos criar softwares de teste perfeito.

Devemos testar os testes.

Após criar um teste para detectar um erro específico, cause o erro deliberadamente e verifique se o teste reclama.

Dicas

DICA 64: Use sabotadores para testar seus testes.

Designe um sabotador de projeto, seu papel será introduzir erros de propósito e verificar se os testes estão os capturando.

Testando Totalmente

Não há uma forma de saber se você testou a base do código com abrangência suficiente, mas há produtos que podem te dar uma noção disso.

Ferramentas de análise de cobertura assistiram o código durante os testes e dirão quais linhas foram executadas.

Mesmo percorrendo todas as linhas do código, ainda há algo mais importante, os estados que seu programa pode ter.

Estados de um programa

```
public int test(int a, int b){
    return a / (a + b);
}
```

Esse código de apenas 3 linhas tem 1.000.000 de estados lógicos, 999.999 dos quais funcionarão corretamente e um que não funcionará (quando a + b é igual a zero). Saber que as linhas foram executadas não te dará noção disso.

Testar todos os estados possíveis é difícil como “o sol será uma massa fria e dura antes que você consiga resolvê-lo”.

Dicas

DICA 65: Teste a cobertura de estados e não a cobertura do código.

Quando testar

Assim que existir algum código de produção, ele deve ser testado, alguns sistemas de controle de código-fonte, como o Aegis, podem realizar essa tarefa automaticamente. Se não, basta executar o comando % make test.

Alguns testes não são fáceis de serem feitos com tanta frequência, ou porque, precisam de configurações e equipamentos especiais, como os testes de resistência, ou porque não podem ser feitos automaticamente. Porém, ainda sim devem ser testados com certa constância, seja semanalmente ou mensalmente.

Reforçando a rede

Caso um erro passar pela sua rede de testes existentes, você terá de adicionar um novo teste para garantir que esse erro não fuja da próxima vez.

Dicas

DICA 66: Encontre os erros apenas uma vez

Uma vez que um testador humano identifica um erro, a abordagem eficaz é modificar os testes automatizados para detectar esse erro específico de forma contínua, sem exceções.

Tudo se resume a escrever

Não deixe a documentação se tornar um elemento de segunda classe, banido do fluxo de trabalho principal do projeto.

"A tinta mais fraca é melhor do que a memória mais afiada."

Provérbio chinês.

A ideia é reduzir a separação entre código e documentação, tratá-los como dois aspectos do mesmo modelo e aplicar princípios pragmáticos a ambas as partes.

Dicas

DICA 67: Trate o português simplesmente como outra linguagem de programação

Dois tipos básicos de documentação

- Documentação interna: inclui os comentários do código-fonte, documentos do projeto e de teste e assim por diante.

- Documentação externa: inclui os comentários do código-fonte, documentos do projeto e de teste e assim por diante.

"*Toda documentação é um espelho do código. Se houver uma discrepância, o código é o que importa – para o bem ou para o mal.*"

Dicas

DICA 68: Construa a documentação no código, não a acrescente como complemento.

Comentários no código

“O código deve ter comentários, mas comentários demais podem ser tão prejudiciais como comentários de menos.”

Comentários bem-vindos

- Autoria e Propriedade
- Documentação de Partes Vagas: esses comentários lhe darão oportunidade para documentar decisões de engenharia, justificativas e alternativas descartadas no projeto.
- Cabeçalho simples no Nível do Módulo.
- Declarações de Dados e Tipos Significativos
- Cabeçalho por Classe e Método: Descrever como a função é usada e qualquer coisa que ela faça, que não seja óbvia.

Autoria

Uma das principais informações que devem estar no arquivo fonte é o nome do autor.

Designar responsabilidade e autoridade pelo código-fonte é crucial para promover honestidade entre os colaboradores.

Pode ser preciso adicionar observações de direitos autorais ou textos legais padronizados em cada arquivo-fonte. Seu editor pode ser configurado para inseri-los automaticamente.

Comentários bem-vindos: Exemplo

Observe a sugestão de comentário do JavaDoc:

```java
/**
 * Encontra o valor de pico (mais alto) dentro de um intervalo de
 * datas de amostras especificado.
 *
 *
 * @param aRange     Intervalo de datas a ser pesquisado.
 * @param aThreshold Valor mínimo a considerar.
 * @return o valor ou <code>null</code> se não for encontrado nenhum
 * valor maior ou igual ao limite.
 */
public Sample findPeak(DateRange aRange, double aThreshold)
```

Nomes de variáveis

Dê nomes significativos.

"*Lembre-se de que você (e outras pessoas depois de você) vai ler o código muitas centenas de vezes, mas só o escreverá algumas. Não tente ganhar tempo escrevendo cp em vez de connectionPool.*"

Abandone a Notação Húngara

A Notação Húngara, aquela em que as informações de tipo da variável são codificadas no próprio nome, é extremamente inapropriada em sistemas orientados a objetos.

Troque isso:

```
private String strName;
private String strLocation;
private int intPhoneExtension;
```

Por isso:

```
private String name;
private String location;
private int phoneExtension;
```

Efeito Stroop: Cuidado

Se há um método chamado capturaDado você espera que ele capture dado. Mas e se na verdade ele gravar o dado?

```
public static void capturaDado(int dado){
    File file = new File("arquivo.txt");

    try {
        FileWriter fileWriter = new FileWriter(file);
        fileWriter.write(dado);
        fileWriter.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
```

Efeito Stroop

O cérebro humano se confunde com isso – o nome disso é efeito Stroop.

Os nomes são muito relevantes para o nosso cérebro.

"Nomes ambíguos levarão ao caos no seu código."

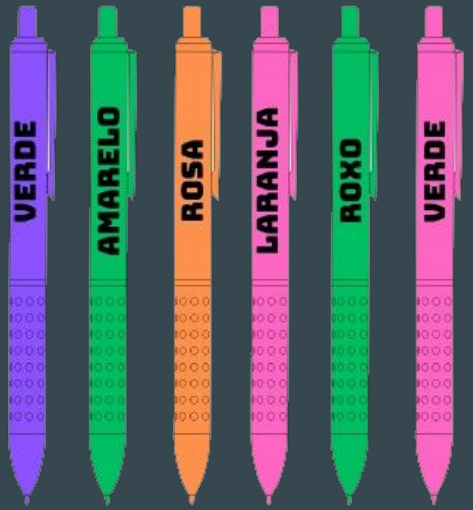
Efeito Stroop: Visualização

Faça o teste:

Reúna algumas canetas coloridas e utilize-as para anotar os nomes das cores, porém não escreva um nome de cor usando uma caneta dessa cor.

Depois de escrever os nomes das cores, diga em voz alta a cor que você escreveu, o mais rápido possível.

Uma hora você irá se confundir!!



Não Deve Aparecer em Comentários:

- Lista das funções exportadas por um código do arquivo, há programas que analisam a fonte para nós, utilize-os.
- Histórico de revisões, há sistemas de controle de código-fonte para isso, porém pode ser interessante informações sobre a data da última alteração e a pessoa que a fez.
- Lista de outros arquivos que esse arquivo usa, pode ser determinado mais precisamente com o uso de ferramentas automáticas.
- O nome do arquivo, o RCS e sistemas semelhantes podem manter essa informação atualizada automaticamente.

Documentos executáveis

Com a definição de comentários significativos, ferramentas como o JavaDoc [URL 7] e o DOC++ [URL 21] têm a capacidade de extrair e formatar esses comentários, permitindo a geração automática de documentação de nível de API.

Nesse caso, o modelo é o código-fonte: uma visualização do modelo pode ser compilada; outras visualizações devem ser impressas ou vistas na Web.

Nosso objetivo é sempre trabalhar sobre o modelo, seja o modelo o próprio código ou algum outro documento, e ter todas as visualizações atualizadas automaticamente.

Por que cumprir esse objetivo?

Suponhamos que tivéssemos uma especificação que listasse as colunas de uma tabela de banco de dados. Também teríamos um conjunto separado de comandos SQL para criar a tabela real no banco de dados e, provavelmente, algum tipo de estrutura de registro de uma linguagem de programação para conter o conteúdo de uma linha da tabela.

O problema aqui é a redundância de informações, que resulta em potenciais inconsistências quando uma dessas três fontes é atualizada. Essa situação viola o princípio NSR.

Ao selecionar uma fonte de informações autorizada, de forma a torná-la o modelo central do processo e exportá-la com formas de visualizações diferentes soluciona-se a problemática.

Como fazer isso?

Se a fonte escolhida for um documento armazenado em texto simples com comandos de marcação (usando HTML, por exemplo), você poderá usar ferramentas como a linguagem Perl para extrair o esquema e formatá-lo automaticamente.

Agora, se ele estiver em um formato binário de processador, você pode

- Criar Macros: Utilize linguagens de macro para programar a exportação de seções específicas do documento. Em casos de dificuldade, exporte a seção para um arquivo de texto simples e use ferramentas como Perl para a conversão.

- Subordinar o documento: adote outra representação e crie uma ferramenta para exportar essa informação para o documento, estabelecendo uma conexão eficaz entre as diferentes formas.

Redatores técnicos

Um problema bastante comum quando há redatores técnicos no projeto, é a falha de comunicação entre os redatores e os programadores.

Normalmente os programadores transferem sem muita colaboração as informações relacionadas ao projeto para os redatores técnicos, o que causa um gap entre os dois cargos.

Isso é um erro, pois sem a orientação correta dos programadores, os redatores podem não aplicar os princípios pragmáticos.

“Só porque os programadores não estão redigindo esses documentos não significa que possamos renunciar aos princípios pragmáticos”.

Imprimir ou publicar na web?

O ideal é tentar produzir toda a documentação de forma que possa ser publicada online, na web, complementada com hiperlinks, em conjunto com um carimbo de data ou um número de versão em cada página da Web.

Dessa forma, o leitor poderá ter uma boa ideia do que está atualizado e do que foi ou não alterado recentemente.

Documentação em outros formatos

Frequentemente, é necessário fornecer a mesma documentação em diversos formatos, como um documento físico, ou até mesmo uma apresentação de slides.

O certo a ser feito é garantir que apresentação de um documento seja independente de seu conteúdo.

Como modificar o formato?

- Se estiver usando um sistema de marcação como HTML:

  Há uma flexibilidade para implementar em quantos formatos de saída forem necessários, a partir dessas marcações ferramentas como XSL e o CSS - eXtensible Style Language e Cascading Style Sheets podem ser usadas para gerar tais saídas.

- Caso esteja utilizando um processador de texto:

  Não deixe de utilizar estilos diferentes para identificar diferentes elementos do documento, a maioria dos processadores de texto permite a conversão do documento para formatos como o HTML para publicação na Web.

Linguagens de Marcação

Sua marcação deve ser suficiente para expressar todos os conceitos necessários, e a conversão para qualquer outra forma publicável será fácil e automática.

Você pode utilizar o DocBook, um padrão de marcação baseado na SGML que identifica cuidadosamente cada componente de um documento.

O DocBook é utilizado no projeto de documentação do Linux.

Grandes Expectativas

"Espantai-vos disto, ó céus, e horrorizai-vos..."

Jeremias 2:12

Dicas

DICA 69: Exceda gentilmente as expectativas de seus usuários

Um projeto que fica abaixo das expectativas está fadado ao fracasso. No entanto, se você exceder muito os limites também falhará.

Comunicando expectativas

Os usuários irão representar alguma visão do que querem, podendo ser incompleta, inconsistente ou tecnicamente impossível.

Parte do seu papel é comunicar as áreas em que as expectativas não poderão ser plenamente cumpridas.

Os usuários devem ter consciência do que lhe será entregue.

Se a equipe se comunicar fluentemente com o ambiente externo esse processo será quase automático

O ganho extra

Cuidado, se você comunicar tudo que está fazendo aos usuários, não haverá nenhuma surpresa no final, isso não é bom.

Surpreenda seus usuários, os encante.

Durante o desenvolvimento, ouça atentamente os usuários, busque pistas sobre recursos que os deixam felizes.

Recursos que trazem felicidade

- Ajuda com balões e dicas de ferramentas
- Atalhos no teclado
- Um breve guia de referência como complemento do manual do usuário
- Uma tela inicial personalizada para a empresa

Atente-se para não danificar o sistema quando adicionar novos recursos

Orgulho e preconceito

Em alguns projetos a ideia de propriedade pode causar problemas.

Não tente defender possessivamente seu código

Trate o código dos outros como gostaria que tratassem o seu, não tenha preconceito de modo a desmerecer o código de seus colaboradores.

Orgulhe-se da sua autoria, faça da sua assinatura um indicador de qualidade.

"Você já nos encantou por tempo suficiente."

Jane Austen, Orgulho e preconceito

Dicas

DICA 70: Assine seu trabalho

Essa é a dica final e representa a consequência direta de todo o resto.

Sobre os autores

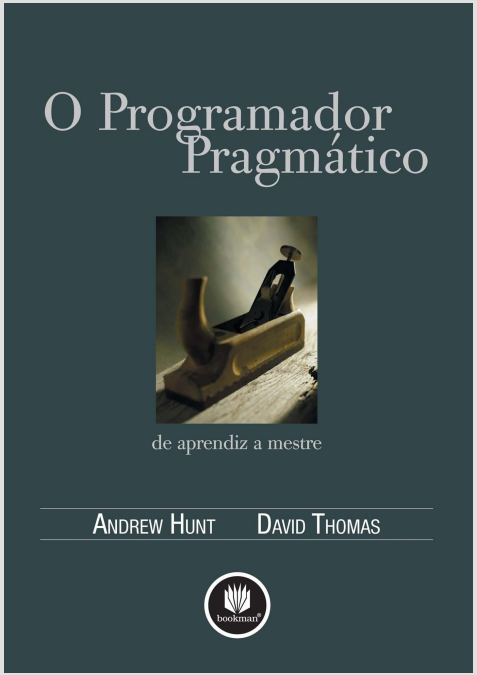
Andrew Hunt trabalhou em diversas áreas, como telecomunicações, serviços financeiros, artes gráficas etc. Hunt se especializou em combinar técnicas já consolidadas com tecnologias de ponta, criando soluções novas e práticas. Ele administra sua empresa de consultoria em Raleigh, Carolina do Norte.

Em 1994 , David Thomas, fundou na Inglaterra uma empresa de criação de software certificada pela ISO9001 que distribuiu mundialmente projetos sofisticados e personalizados. Hoje, Thomas é consultor independente e vive em Dallas, Texas.

Atualmente, David e Andrew trabalham juntos em The Pragmatic Programmers, L.L.C

Este conjunto de slides foi elaborado a partir da obra:

HUNT, Andrew ; THOMAS, David. **O Programador Pragmático:** de aprendiz a mestre.Bookman, 2010.



A equipe

Ana Lívia Motta, Autora

Aluna e bolsista do PET/ADS desde julho de 2023.
LinkedIn

Lucas Oliveira, Tutor

Professor de Computação, é tutor do PET/ADS desde janeiro de 2023.
LinkedIn