O Programador Pragmático:

De Aprendiz a Mestre

•••

Capítulo 3: As Ferramentas Básicas

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

Introdução

Assim como o marceneiro possui martelos, serrotes, tornos e brocas, um programador também precisa de ferramentas práticas e duráveis para auxiliá-lo.

Esta apresentação discute a importância de investir em sua própria caixa de ferramentas, como elas moldam a maneira com que você trabalha e como você moldá-las em uma extensão de si mesmo.

Como programadores pragmáticos, nosso material básico para modelar não é madeira ou ferro, mas informações.

Existem muitas formas de armazenar informações, mas idealmente como podemos armazená-las para que sejam de fácil acesso e edição?

Dicas

DICA 20:

Mantenha as informações em texto simples

Por sua persistência, praticidade e facilidade de leitura, é a melhor maneira de armazenar dados.

O poder do texto simples

O texto simples nada mais é que uma sequência de caracteres imprimíveis que podem ser lidos e entendidos diretamente pelas pessoas.

Por exemplo, embora o trecho a seguir seja composto por caracteres imprimíveis, ele não tem significado.

Field19=467abe

Não é possível saber o que ele representa. Uma opção melhor seria deixá-lo inteligível:

DrawingType=UMLActivityDrawing

O poder do texto simples - Desvantagens

Há duas principais desvantagens ao utilizar o texto simples:

- Armazenamento: pode ocupar mais espaço do que um formato binário compactado;
- Poder de Processamento: pode ser computacionalmente mais caro interpretar e processar um arquivo de texto simples.

Dependendo de seu aplicativo, essas situações podem ser inaceitáveis, veja dois exemplos:

- Armazenamento de dados de telemetria por satélite;
- Formato interno de um banco de dados relacional.

O poder do texto simples - Desvantagens?

Nessas situações, pode ser aceitável armazenar metadados sobre os dados brutos em texto simples.

Hoje em dia não se pensa muito sobre ficar sem armazenamento ou ter pouco processamento, por conta do avanço dos computadores modernos.

E o formato binário vem com seu próprio conjunto de problemas, que além de separar os dados de seu significado, necessitando de um contexto explícito, não é mais seguro que o texto simples.

Texto simples ou codificação binária?

Digamos que você queira armazenar uma propriedade do seu programa. Com texto simples, ela poderia ser escrita como algo assim:

myprop.uses_menus=FALSE

Agora compare com sua versão em binário:

0010010101110101

Com texto simples você pode obter um fluxo de dados autodescritivo que seja independente do aplicativo que o criou, muito mais customizável e com certeza mais legível.

Texto simples ou codificação binária

Desenvolvedores podem preferir os dados em binário por achar que, ao salvar metadados em texto simples, estarão expondo-os aos usuários do sistema.

Esse receio é infundado, eles podem ser mais obscuros que texto simples, mas não são mais seguros.

Se a segurança das informações é algo importante, o certo seria criptografá-las com um hash seguro.

O poder do texto simples - Vantagens

Já sabemos que as desvantagens podem ser mitigadas, mas quais são as reais vantagens de se usar o texto simples?

- Garantia contra a obsolescência.
- Aproveitamento universal.
- Facilidade de testar.

Garantia contra a obsolescência

Imagine que você precisa extrair uma lista de cpfs em um sistema legado e, no arquivo de dados, você encontra o seguinte formato:

<CPF>123456789-11</CPF> …<CPF>567345098-11</CPF>

Mesmo sem as características do arquivo, você poderia facilmente criar um programa para extrair esses dados. Mas, suponhamos que, na verdade eles estivessem armazenados assim:

AX0L12345678911K8&X … LY9T567345098119 9KI

É nítida a diferença entre *legível para humanos* e *compreensível para humanos*.

Garantia contra a obsolescência

Desde que os dados sobrevivam, você terá a chance de usá-los em outros projetos, mantendo a utilidade deles mesmo depois que o projeto original se extinguiu.

Isso não poderia ser atingido em outras formas de armazenamento que necessitam de contexto, como arquivos binários.

“Formas de dados legíveis por humanos e dados autodescritivos sobreviverão a todas as outras formas de dados e aos aplicativos que as criaram. Ponto.”

Aproveitamento universal

Praticamente qualquer ferramenta digital pode operar com o texto simples, e alguns sistemas são pensados para armazenar todos os seus arquivos dessa forma, como o Unix.

Caso uma falha ou necessidade de configuração ocorra, não é mais necessário reinstalar o sistema inteiro ou utilizar ferramentas complexas para modificá-lo, agilizando imensamente o trabalho. O mesmo pode até ser mantido em um sistema de controle de versão (CVS) para facilitar ainda mais o processo.

Facilidade de testes

O texto simples facilita também o manejo de dados usados em testes:

- Não é necessária nenhuma ferramenta especial na criação de dados sintéticos de entrada, os tornando livremente modificáveis antes do seu uso em testes.
- Valores de saída em texto simples podem ser trivialmente analisados com ferramentas do sistema, e também utilizando linguagens de *scripting* para maior análise.

Jogos de shell

Utilizar um ambiente integrado de desenvolvimento (IDE) pode trazer consigo várias facilidades, mas traz também muitas limitações que podem estagnar as potencialidades de um desenvolvedor:

Um verdadeiro programador pragmático deseja estar presente em todo o processo do código, assim ele não pode ficar preso a apenas o que a ferramenta foi feita para fazer.

O console é muito mais poderoso que qualquer IDE no quesito de agilidade e extensibilidade, se tornando um verdadeiro *playground* para o programador.

Dicas

DICA 21:

Use o poder dos shells de comando.

Faça testes com seu shell de comando e ficará surpreso com quanto ele lhe tornará mais produtivo!

Jogos de shell

Se você não está acostumado, utilizar os comandos pode parecer um pouco assustador, mas veja neste exemplo como ele pode te ajudar a ganhar tempo.

Como descobrir quais arquivos Java não foram alterados na última semana:

Com Shell..

```
find . -name '*.java' -mtime +7 -print
```

Com GUI..

Clique e navegue em "Find files", clique no campo "Named" e digite ".java", selecione a guia "Date Modified". Em seguida, selecione "Between". Clique na data inicial e digite a data inicial do começo do projeto. Clique na data final e digite a data de uma semana atrás a partir de hoje (certifique-se de ter um calendário à mão). Clique em "Find Now".*

Jogos de shell

Os utilitários de shell do Windows não são tão poderosos quanto os do sistema Linux, mas não é por isso que eles são inúteis!

Ainda é possível utilizar a *programação em batch* e o mais recente PowerShell, no Windows, para pensar fora da caixa e criar automatizações.

Também é possível utilizar programas para compatibilizar o sistema com comandos do Linux, como o **Cygwin** e o **UWIN**.

Edição avançada

Com o rápido avanço das ferramentas digitais, entramos numa era em que escolher um editor para o nosso código não é mais uma tarefa fácil.

Há centenas de milhares deles, com diversas funcionalidades e atalhos diferentes. Então como podemos tomar uma decisão concreta?

Dicas

DICA 22:

Use um único editor bem

Utilize apenas um editor para a maioria do que você precisar, e pratique o suficiente para conhecê-lo de dentro a fora.

Edição avançada

Sem um editor exclusivo você fica realmente perdido, e pode acabar usando múltiplos, um para cada coisa que você faz, o que te torna proficiente em nenhum.

Por isso é melhor que se utilize majoritariamente um, e o utilize para todas as suas tarefas de edição.

Dessa maneira você irá manipular texto sem precisar parar para pensar, e seu trabalho fluirá muito melhor.

Sabendo disso, devemos analisar as características de cada editor para fazer uma única escolha.

Recursos do editor

Além de levar em conta suas preferências pessoais, existem recursos que todo bom editor de texto deve possuir:

- Configurabilidade: todos os aspectos do editor devem ser configuráveis de acordo com suas preferências, cores, fontes, atalhos e etc.
- Extensibilidade: ele não pode se tornar obsoleto com o surgimento de novas tecnologias, devendo ser capaz de integrar novas linguagens, ambientes de compilação, e suas nuances.
- Programabilidade: deve ser possível programar o editor para executar tarefas complexas.

Se seu editor atual for deficiente em alguma dessas áreas, talvez seja hora de considerar um novo e mais avançado.

Recursos do editor

É bom também realçar aqueles que dão suporte a recursos específicos a determinadas linguagens de programação:

- Realce de sintaxe
- Autoconclusão (autocomplete)
- Autorrecuo
- Texto padronizado inicial de código ou documento
- Conexão com recursos de ajuda
- Recursos como os dos IDE (compilação, depuração e etc)

Produtividade

Só utilizar os recursos de um editor cegamente, recortando e colando com o mouse, digitando todas as linhas manualmente, não o torna é diferente de um bloco de notas.

Explore as funcionalidades e as customize para maximizar seu conforto e produtividade, automatizando configurações de rotina com scripts e aperfeiçoando sua navegação com macros.

Para onde ir daqui

Se isso tem a sua cara...

“Só uso recursos básicos de muitos editores diferentes.”

“Tenho um editor favorito, mas não uso todos os seus recursos.”

“Tenho um editor favorito e uso-o onde possível.”

“Só uso e sempre usei o bloco de notas, e nunca vou parar !!!!!!!”

Então tente...

Selecionar um editor poderoso e aprender a usá-lo com destreza.

Aprender a usá-los. Reduza a quantidade de teclas que você tem de pressionar.

Expandir isso e usar para mais tarefas do que já usa.

Sem problemas! Mas se começar a sentir inveja de outros editores, não hesite em testá-los. :)

Controle de código-fonte

Durante o desenvolvimento de um projeto, com certeza o atalho $ctrl + z$ é o mais utilizado dentre as combinações de tecla, pois ele reduz a dor de cabeça com pequenos erros.

Isso é extremamente útil em um escopo contido, mas, e se ao invés de retornar algumas linhas, fosse necessário retornar scripts inteiros ao seu estado de dias, ou até semanas atrás?

A resposta é utilizar um Sistema de Controle de Versão!

Dicas

DICA 23:

Use sempre o controle do código-fonte

Use mesmo se aquilo em que estiver trabalhando não for código-fonte.

Controle de código-fonte

A funcionalidade de um VCS é registrar cada alteração feita no projeto e sua documentação, mas as vantagens vão muito além de apenas catalogar mudanças.

Com o VCS é possível ter informações sobre as alterações e os envolvidos, bem como o número de linhas modificadas e a pessoa que as modificou.

O VCS também permite o desenvolvimento concorrente de duas ou mais pessoas em um mesmo projeto.

O controle do código-fonte e as construções

Uma outra vantagem de se usar um VCS é automatizar a construção de um projeto para que ela seja segura e repetível.

Uma automação pode extrair o código fonte mais recente do projeto e executar testes de regressão para verificar se não há nada quebrado depois do trabalho diário.

Automações asseguraram a consistência de maneira orgânica e sem necessidade de colocar a mão na massa repetidamente.

“M-mas minha equipe não está usando um VCS!”

Isso soa como uma ótima oportunidade para demonstrar as maravilhas que um sistema de gerenciamento de versão pode trazer para o seu trabalho!

Você não precisa ter pressa ou aborrecer as pessoas para que isso ocorra, comece configurando um para você mesmo.

Lembra da sopa de pedras? Você verá como as pessoas vão se mostrar interessadas naturalmente ao ver as facilidades em prática.

Depuração e sua psicologia

Nenhum software pode ser à prova de falhas, sendo assim, grande parte do desenvolvimento é gasto na depuração, sendo esta:

“O processo da descoberta de erros no programa e a resolução dos mesmos.”

Muitos desenvolvedores veem esse tópico como dor de cabeça, ao invés de ser encarado como um quebra-cabeça a ser solucionado, ele é visto como motivo para:

- Criar brigas
- “Se livrar” da culpa de ter feito um código falho.

Dicas

DICA 24:

Corrija o problema, esqueça o culpado

Não importa se você ou outra pessoa foi o culpado pelo bug, ele continuará sendo seu problema.

Uma mentalidade para a depuração

“Aceite o fato de que a depuração é apenas resolução de problemas e encare-a como tal.”

Deixe de lado o ego e saiba que todos estão no time para resolver um problema em conjunto.

Mesmo com uma boa convivência em equipe, alguns fatores podem atrapalhar a deputação:

- Chefe irritado
- Cobranças excessivas
- Falta de confiança

Nesses momentos, é sempre bom ter a cabeça fria e pensar com cuidado.

Dicas

DICA 25:

Não entre em pânico

É fácil entrar em pânico, mas você deve combater as preocupações externas e internas para conseguir testar o código confortavelmente.

Uma mentalidade para a depuração

O que importa é conseguir manter a calma para raciocinar sobre o problema que está ocorrendo:

- Não se prenda a pensamentos paralisantes como “*Isso não tem como acontecer*”, afinal aquilo já está acontecendo, e sempre há uma causa por trás.
- Resista ao sentimento de corrigir apenas os sintomas visíveis, é provável que o erro esteja longe do observável e pode envolver outras funções relacionadas.
- Tente sempre descobrir a causa raiz de um problema e não apenas um aspecto específico dele.

Onde começar

1. Certifique-se de estar trabalhando em um código que tenha sido totalmente compilado e sem avisos.

   Não faz sentido perder tempo com erros que o próprio compilador pode resolver.

2. Comece a coletar dados sobre o bug.

   Geralmente relatos não são muito precisos, pode ser necessário observar o usuário que relatou para obter mais detalhes.

3. Teste exaustivamente tanto condições *limítrofes* quanto padrões de uso realistas do usuário final.

   Testes *artificiais* não exercitam um projeto suficientemente.

Estratégias de depuração - Reprodução de bugs

O melhor jeito de corrigir um bug é torná-lo **facilmente reprodutível**.

Se conseguir meios de sempre desencadear um bug, você terá um entendimento muito maior do que o causa e como é possível evitar casos parecidos no futuro usando testes.

Mas isso não adianta se são necessários 15 passos para reproduzi-lo. Ao forçar o isolamento das circunstâncias que exibem o bug, podemos obter uma percepção ainda maior de como corrigi-lo.

Estratégias de depuração - Visualize seus dados

Uma ótima maneira de entender o funcionamento de um programa é ver quais dados ele está operando.

Geralmente os programadores imprimem dados de depuração no console, mas é possível ter uma percepção muito maior deles ao utilizar um depurador.

Mesmo se seu depurador possuir suporte limitado à visualização, você ainda pode fazer isso manualmente, com papel e lápis ou com programas de plotagem externos.

Estratégias de depuração - Rastreando

Ferramentas de depuração podem nos mostrar o estado imediato de um programa, mas as vezes é necessário o verificar durante várias etapas da execução. Como em sistemas de tempo real.

As *instruções de rastreamento* são mensagens curtas que se envia para o console ou registra em um arquivo a fim de testar o código, como “cheguei aki” e “x = 5”.

- Podem rastrear erros que os depuradores não conseguem.
- Inestimável em sistemas que tem o tempo como fator.

Contudo, devem estar sob um formato consistente e regular, podendo ser sujeitas a análises mais avançada e automática em arquivos de log.

Estratégias de depuração - Rubber ducking

Rubber Ducking é uma técnica simples, mas extremamente efetiva: explique para alguém o passo a passo de como o programa está funcionando.

A pessoa não precisa falar nada, o simples ato de explicar sua linha de raciocínio faz o problema se evidenciar, pois força o programador a explicitar suas decisões e olhar de outra perspectiva.

Se não houver um parceiro humano, patos de borracha também funcionam excepcionalmente bem.

Estratégias de depuração - Processo de eliminação

“Trabalhamos em um projeto em que um engenheiro sênior estava convencido de que a chamada de sistema Select estava com defeito no Solaris. Não havia elementos de persuasão ou lógica suficientes que o fizessem mudar de ideia (o fato de que todos os outros aplicativos de rede da máquina estavam funcionando bem era irrelevante).

Ele passou semanas criando soluções, que, por alguma razão desconhecida, não pareciam corrigir o problema. Quando finalmente se viu forçado a se sentar e ler a documentação sobre select, descobriu o problema e o corrigiu em questão de minutos.”

Dicas

DICA 26:

“O select não está com defeito”

A chance de um erro existir no compilador, sistema operacional ou biblioteca e estar interferindo com o seu projeto, é pequena demais para ser considerada.

Estratégias de depuração - Processo de eliminação

Existe uma chance muito pequena de existir um bug no compilador, sistema operacional ou biblioteca interferindo com o seu projeto.

É **muito** mais provável que o bug esteja no aplicativo em desenvolvimento.

"Se encontrar pegadas feitas por cascos, pense em cavalos – não em zebras."

Estratégias de depuração - Processo de eliminação

Se não houver um local óbvio para começar a depurar, pense como uma **busca binária**:

1. Verifique se o erro está acontecendo em dois locais distantes no código.
2. Analise do começo para o meio e depois do meio para o fim.
3. Repita e entre em funções mais específicas dentro do programa.

Assim você poderá estreitar a localização do bug até encontrá-lo!

Estratégias de depuração - O elemento surpresa

Muitas vezes você pode ser surpreendido por um bug, mesmo tendo certeza absoluta de que seu programa está funcionando.

Nessas horas devemos deixar de lado nossas “verdades” sobre o código que escrevemos e perceber que provavelmente estamos errados em nossas suposições.

Não se deve ignorar uma rotina ou um bloco de código envolvido em um bug só porque se “sabe” que ele funciona.

Dicas

DICA 27:

Não suponha, teste

Além de corrigir o bug, você deve determinar por que essa falha não foi detectada antes.

Estratégias de depuração - O elemento surpresa

Surpresas podem ser contidas para que não voltem a acontecer:

- Considere aperfeiçoar os testes existentes para que eles possam detectar o erro e similares.
- Se o bug foi causado por dados inválidos, propagados em vários níveis antes, veja se uma melhor verificação desses valores em rotinas poderia o ter impedido de executar o erro.
- Busque por outros locais no código que podem ser propensos a falhar do mesmo jeito.

Estratégias de depuração - O elemento surpresa

Surpresas podem ser contidas para que não voltem a acontecer:

- Se o processo de corrigir o bug for demorado, tente fazer testes com uma maior cobertura, ou também crie um *analisador de arquivos de log*.
- Se o bug foi causado pela suposição errada de alguém, discuta o problema com a equipe. Se alguém entendeu errado, outros também podem.

Lista de verificação da depuração

Se faça algumas perguntas para chegar a raiz do problema e entendê-lo:

- O problema relatado é resultado de um bug subjacente ou um sintoma?
- Em que parte específica do sistema o bug se encontra?
- Como você explicaria detalhadamente este problema a um colaborador?
- As condições que causaram esse bug estão presentes em outros lugares do sistema?
- Se o código suspeito passou por testes de unidade, será que eles estão realmente completos?

Manipulação de texto

Um programador pragmático deve possuir no seu arsenal um meio para editar texto e trabalhar adaptativamente com ele, conseguindo modelar fluidamente e encaixá-lo onde precisar.

Esse meio deve ser uma ferramenta de uso geral, como linguagens de manipulação de texto (Python, Perl, etc).

Essas são tecnologias importantes para a capacitação do programador. Com elas você pode criar utilitários e protótipos de ideias rapidamente.

Dicas

DICA 28:

Aprenda uma linguagem de manipulação de texto

Essas linguagens podem ser extensas e levar tempo para se dominar, mas em mãos-hábeis se tornam poderosas e versáteis.

Manipulação de texto - Utilizações

Exemplos de alguns usos possíveis das linguagens de uso geral:

- Manutenção de um *schema* de banco de dados
- Gerar documentação para aplicações web
- Configuração de propriedades Java
- Interface entre C e Object Pascal
- Geração de dados de teste
- Redação de livros

Geradores de código

Comumente, programadores se veem em situações onde precisam da mesma informação ou estrutura em diferentes contextos.

Assim como os marceneiros confeccionam gabaritos para facilitar o trabalho, programadores podem construir geradores de código.

Dicas

DICA 29:

Escreva um código que crie códigos

Uma vez construído, ele pode ser usado durante toda a vida do projeto a praticamente nenhum custo.

Geradores de código

Há dois tipos principais:

1. Geradores de código passivos: são executados apenas uma vez para produzir um resultado, que se torna independente do gerador.
2. Geradores de código ativos: são executados sempre que seus resultados forem necessários.

Nos geradores de código ativos, os resultados são descartáveis, pois podem ser gerados novamente por demanda.

Geradores de código passivos

Funcionam como geradores de modelo por parâmetros, e possuem muitas utilidades:

- Criação de novos arquivos-fonte com configurações default, indicadas pela necessidade do programador.
- Execução de conversões exclusivas entre linguagens de programação. Elas não precisam ser 100% precisas, apenas agilizar o trabalho.
- Produzir ou calcular dados de consulta que dificultariam o processamento em tempo de execução.

Geradores de código ativos

Geradores ativos são extremamente úteis quando se precisa fazer dois ambientes diferentes funcionarem em conjunto.

Por exemplo, se quer ligar um banco de dados ao desenvolvimento:

- Crie um gerador que transforma esquemas em estruturas de outra linguagem, funciona muito melhor que apenas duplicar informações.

Geradores de código não precisam ser complexos

O nome passa a impressão de ser uma ferramenta complexa, mas normalmente a parte mais cabeluda é o analisador:

- Transforma entrada na saída desejada.

A chave para diminuir a complexidade é **manter a entrada simples**, que ocasionará em um gerador simples.

E nem gerar código!

Você pode utilizar geradores para confeccionar qualquer tipo de dado que quiser:

- Texto simples
- HTML
- XML
- Markdown
- Qualquer texto que possa ser uma entrada para algum local do projeto

Sobre os autores

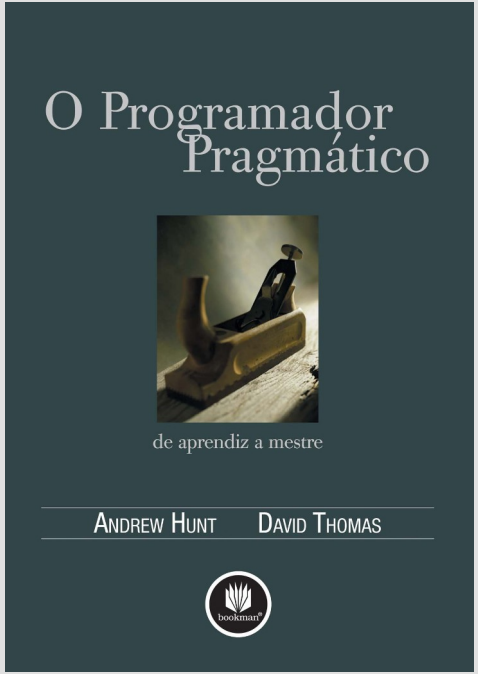
Andrew Hunt trabalhou em diversas áreas, como telecomunicações, serviços financeiros, artes gráficas etc. Hunt se especializou em combinar técnicas já consolidadas com tecnologias de ponta, criando soluções novas e práticas. Ele administra sua empresa de consultoria em Raleigh, Carolina do Norte.

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

Vitor Bonelli, Autor

Aluno e bolsista do PET/ADS
LinkedIn

Lucas Oliveira, Revisor

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