13. Caráter Pessoal

O caráter pessoal tem recebido um escasso grau de atenção no desenvolvimento de software. Desde o histórico artigo de 1965 de Edsger Dijkstra, *"Programming Considered as a Human Activity"* (Programação Como uma Atividade Humana), o caráter do programador tem sido visto como uma área de investigação legítima e fértil. Títulos como "A Psicologia da Construção de Pontes" e "Experiências Exploratórias no Comportamento do Advogado" poderiam parecer absurdos, mas no campo da computação, "A Psicologia da Programação de Computadores", "Experiências Exploratórias no Comportamento do Programador" e títulos semelhantes são clássicos.

Engenheiros de todas as disciplinas aprendem os limites das ferramentas e dos materiais com que trabalham. Se você é engenheiro eletricista, conhece a condutividade de vários metais e mil maneiras de usar um voltímetro. Se você é engenheiro de estruturas, conhece as propriedades de sustentação da madeira, do concreto e do aço.

Se for engenheiro de software, seu material de construção básico é o intelecto humano, e sua principal ferramenta é você. Em vez de projetar uma estrutura com todos os detalhes e depois levar as plantas para alguém construir, você sabe que, quando tiver projetado um software até o último detalhe, ele estará pronto. O trabalho da programação é mesmo construir castelos no ar - é uma das atividades mais puramente mentais que você pode exercer.

Consequentemente, quando os engenheiros de software estudam as propriedades essenciais de suas ferramentas e matérias primas, eles descobrem que estão estudando as pessoas: o intelecto, o caráter e outros atributos menos palpáveis do que a madeira, o concreto e o aço.

13.1 O caráter pessoal não é algo fora do tema?

A forte natureza introspectiva da programação torna o caráter pessoal particularmente importante. Você sabe como é difícil passar oito horas por dia concentrado. Você provavelmente já teve a experiência de num certo dia estar esgotado pela alta concentração exigida no dia anterior, ou de estar esgotado em um determinado mês pela alta concentração exigida no mês anterior. Provavelmente, você já teve dias em que trabalhou bem das 8 da manhã às 2 da tarde, e teve a sensação de que cumpriu seu dever. Contudo, você não parou; você continuou das 2 às 5 horas da tarde; depois, passou o resto da semana corrigindo o que escreveu nesse período. Basicamente, o trabalho de programação não é supervisionado, pois ninguém jamais sabe realmente no que você está trabalhando. Todos já tivemos projetos nos quais passamos 80% do tempo trabalhando em uma pequena parte do programa que consideramos interessante e 20% do tempo construindo os outros 80% dele.

Seu chefe não pode obrigá-lo a ser um bom programador; muitas vezes, ele nem mesmo tem condições de julgar se você é bom. Mas se você quiser ser excelente, saiba que a responsabilidade é sua. É uma questão de caráter pessoal.

Uma vez que você tenha decidido tornar-se um programador superior, o potencial de aprimoramento é enorme. Estudos após estudos têm encontrado diferenças na ordem de 10 para 1 no tempo exigido para criar um programa. Eles também têm revelado diferenças na ordem de 10 para 1 no tempo exigido para depurar um programa, e de 10 para 1 no tamanho resultante, na velocidade, na taxa de erros e no número de erros detectados.

Você não pode fazer nada a respeito de sua inteligência, assim diz a sabedoria clássica, mas pode fazer algo sobre seu caráter. E verifica-se que o caráter é o fator

mais decisivo na constituição de um programador superior.

13.2 Inteligência e humildade

A inteligência não parece ser um aspecto do caráter pessoal - e de fato não é. Coincidentemente, uma inteligência brilhante está pouco ligada ao fato de sermos bons programadores.

O quê? Você não precisa ser superinteligente?

Não, não precisa. Ninguém é realmente sábio o bastante para programar computadores. Entender completamente um programa médio exige uma capacidade quase ilimitada de absorver detalhes, e uma capacidade igual para compreendê-los, tudo ao mesmo tempo. A maneira como você focaliza sua inteligência é mais importante do que o nível de inteligência que possui.

Em 1972, Edsger Dijkstra divulgou um artigo intitulado *"The Humble Programmer"* (O Programador Modesto). Ele mostrou que a maior parte da programação é uma tentativa de compensar o tamanho rigorosamente limitado de nossos cérebros. As pessoas que mais se destacam em programação são aquelas que se dão conta de como seus cérebros são pequenos. Elas são modestas. As pessoas que têm o pior aproveitamento em programação são aquelas que se recusam a aceitar o fato de que seus cérebros não estão capacitados para a incumbência. Seus egos as impedem de serem bons programadores. Quanto mais você aprender a compensar seu pequeno cérebro, melhor programador será. Quanto mais modesto você for, mais rapidamente irá se aprimorar.

O objetivo de muitas boas práticas de programação é reduzir a carga sobre sua massa cinzenta. Aqui estão alguns exemplos:

- O objetivo de "decompor" um sistema é torná-lo mais simples de entender.
- Realizar revisões, inspeções e testes é uma maneira de compensar as falibilidades humanas precedentes. Essas técnicas de revisão se originaram como parte da "programação sem ego" (Weinberg 1998). Se você nunca cometesse erros, não precisaria revisar seu software. Mas você sabe que sua capacidade intelectual é limitada; portanto, você a amplia unindo-a à de outra pessoa.
- Manter as rotinas curtas reduz a carga em seu cérebro.
- Escrever programas em termos do domínio do problema, em vez de escrever em termos dos detalhes da implementação de baixo nível, reduz sua carga de trabalho mental.
- Usar convenções de todos os tipos libera seu cérebro dos aspectos relativamente mundanos da programação, os quais oferecem pouco retorno.

Você poderia pensar que o melhor caminho seria desenvolver mais suas faculdades mentais para não precisar dessas muletas da programação. Você poderia pensar que um programador que usa muletas mentais está seguindo pelo pior caminho. Empiricamente, entretanto, tem-se verificado que os programadores modestos, aqueles que compensam suas falibilidades, escrevem código mais fácil para eles mesmos e para outras pessoas entenderem, e com menos erros. O pior caminho, na verdade, é o caminho dos erros e dos cronogramas atrasados.

13.3 Curiosidade

Quando admite que seu cérebro é pequeno demais para entender a maioria dos programas e percebe que a programação eficiente é uma busca por maneiras de compensar esse fato, você começa uma longa procura de maneiras de se superar. No desenvolvimento de um programador superior, a curiosidade sobre assuntos técnicos deve ser uma

prioridade. As informações técnicas relevantes mudam continuamente. Muitos programadores da Web jamais tiveram que programar no Microsoft Windows, e muitos programadores de Windows nunca tiveram que lidar com DOS, UNIX ou cartões perfurados. Os recursos específicos do ambiente técnico muda a cada 5 a 10 anos. Se você não for curioso o bastante para acompanhar as mudanças, poderá se encontrar abatido, jogando cartas na casa de antigos programadores, com o tiranossauro-rex e as irmãs brontossauro.

Os programadores ficam tão ocupados trabalhando que, frequentemente, não têm tempo para serem curiosos a respeito de como poderiam fazer melhor seus trabalhos. Se isso ocorre com você, saiba que não está sozinho. As subseções a seguir descrevem algumas ações específicas que você pode executar para exercitar sua curiosidade e tornar seu aprendizado prioritário.

Construa seu conhecimento a partir do processo de desenvolvimento Quanto mais você souber sobre o processo de desenvolvimento, seja por meio de leitura ou de suas próprias observações sobre desenvolvimento de software, melhor será a posição em que estará para entender as mudanças e para encaminhar seu grupo para uma boa direção.

Se a sua carga de trabalho consiste inteiramente em atribuições de curto prazo que não desenvolvem suas habilidades, demonstre indignação. Se você trabalha em um mercado de software competitivo, metade do que precisa saber agora para fazer seu trabalho estará obsoleto em três anos. Se você não estiver aprendendo, estará se tornando um dinossauro.

Você está sendo requisitado demais para perder tempo ao trabalhar para um gerente que não leva em conta seus interesses. A despeito de alguns altos e baixos, e de alguns trabalhos estarem sendo feitos em outros países, a expectativa é de que o número médio de empregos em software disponíveis aumente substancialmente. Espera-se que os trabalhos para analistas de sistemas aumentem em cerca de 60% e, para engenheiros de software, em cerca de 50%. Para todas as categorias de trabalho com computadores combinadas, cerca de 1 milhão de novos empregos serão criados, além dos 3 milhões já existentes. Se você está sem condições de aprender em seu emprego, procure um novo.

Experimente Uma maneira eficaz de ganhar conhecimento em programação é fazer experiências com ela e com o processo de desenvolvimento. Se você não sabe como funciona um recurso de sua linguagem, escreva um programa curto para testá-lo e ver como ele funciona. Faça um protótipo! Observe o programa ser executado no depurador. É melhor trabalhar com um programa curto para testar um conceito do que escrever um programa maior com um recurso que você não entende direito.

E se o programa curto mostrar que o recurso não funciona da maneira que você desejava? É isso que você queria descobrir. É mais produtivo descobrir isso em um programa pequeno do que em um grande. Um segredo da programação eficiente é aprender a cometer erros rapidamente, aprendendo com eles a cada vez. Cometer um erro não é pecado. Deixar de aprender com um erro, sim.

Leia sobre solução de problemas Solucionar um problema é a atividade básica na construção de software de computador. Herbert Simon relatou uma série de experiências sobre a solução de problemas humanos. Ele descobriu que os seres humanos nem sempre descobrem sozinhos as estratégias de solução de problemas intricados, mesmo que as mesmas estratégias possam ser prontamente ensinadas para as mesmas pessoas (Simon 1996). A implicação é que, mesmo que você queira reinventar a roda, não poderá contar com o sucesso. Em vez disso, você poderia reinventar o quadrado.

Analise e planeje antes de agir Você verá que há uma tensão entre análise e ação. Em algum ponto, você tem de parar de reunir dados e agir. No entanto, para a maioria dos programadores, o problema não é o excesso de análise. Até aqui, o pêndulo está no lado da "ação" do arco e você pode esperar até que ele esteja pelo menos parcialmente no meio, antes de se preocupar com o fato de ficar travado no lado da "paralisia da análise".

Aprenda examinando projetos bem-sucedidos Uma excelente maneira de aprender sobre programação é examinar o trabalho de ótimos programadores. Jon Bentley crê que você deve sentar-se com um cálice de conhaque e um bom cigarro, e então ler um bom programa como leria um bom romance. Isso pode não ser tão exagerado quanto parece. A maioria das pessoas não desejaria aproveitar seu tempo de lazer para examinar minuciosamente uma listagem de código-fonte de 500 páginas, mas muitos gostariam de estudar um projeto de alto nível e de se aprofundar nas listagens mais detalhadas voltadas a áreas selecionadas.

O setor da engenharia de software faz um uso extraordinariamente limitado de exemplos de sucessos e falhas do passado. Se você estivesse interessado em arquitetura, estudaria os desenhos de Louis Sullivan, Frank Lloyd Wright e I. M. Pei. Provavelmente, visitaria alguns de seus prédios. Se estivesse interessado em engenharia estrutural, estudaria a estrutura da ponte do Brooklyn, da ponte Tacoma Narrows e de uma variedade de outras estruturas de concreto, aço e madeira. Você estudaria exemplos de sucessos e falhas em seu campo.

Thomas Kuhn destaca que uma parte de qualquer ciência amadurecida é um conjunto de problemas resolvidos, comumente reconhecidos como exemplos de bom trabalho no setor e que servem como exemplos para trabalhos futuros (Kuhn 1996). A engenharia de software está apenas começando a amadurecer a esse nível. Em 1990, o Computer Science and Technology Board concluiu que havia poucos estudos de caso documentados de seus sucessos ou falhas no setor de software (CSTB 1990).

Um artigo na Communications of the ACM discutiu sobre o aprendizado a partir de estudos de caso de problemas de programação (Linn e Clancy 1992). O fato de que alguém tenha discutido isso é significativo. Também é sugestivo o fato de a mais popular das colunas sobre computação, "Programming Pearls", ter sido elaborada em torno de estudos de caso de problemas de programação. Um dos livros mais populares sobre engenharia de software é *The Mythical Man-Month*, um estudo de caso sobre gerenciamento de programação do projeto OS/360 da IBM.

Com ou sem um livro de estudos de caso sobre programação, encontre um código escrito por programadores superiores e leia-o. Peça para ver o código de programadores que você respeita. Peça para ver o código de programadores que você não respeita. Examine bem ambos os códigos e compare-os com o seu. Quais são as diferenças? Por que eles são diferentes? Qual é a maneira melhor? Por quê?

Além de ler o código de outras pessoas, manifeste o desejo de saber o que os programadores especialistas pensam a respeito de seu código. Encontre programadores de primeira classe que possam elaborar uma crítica sobre o seu trabalho. Após conhecer o teor da crítica, filtre os pontos que estejam relacionados com as idiossincrasias deles e concentre-se nos pontos que importam. Então, altere sua programação de modo que ela fique melhor.

Leia! A fobia da documentação é desmedida entre os programadores. A documentação de computador tende a ser mal escrita e mal organizada, mas apesar de todos os problemas, há muito a ganhar por superar um medo excessivo dos fótons da tela do computador ou de produtos de papel. A documentação contém as chaves do castelo e vale a pena dedicar parte de seu tempo a essa leitura. Ignorar informações que estão

prontamente disponíveis é um descuido tão comum que um acrônimo conhecido nos *newsgroups* e *bulletin boards* é "RTFM!", que significa "Read the!#*%*@ Manual!" (Leia a!#*%*@ do manual!).

Um produto de linguagem moderno normalmente vem acompanhado de um enorme conjunto de códigos de biblioteca. O tempo que se passa examinando a documentação da biblioteca é bem investido. Frequentemente, a empresa que fornece o produto de linguagem já criou muitas classes que você precisa. Nesse caso, procure conhecê-las. Examine a documentação a cada dois meses.

Leia outros livros e periódicos Congratule-se consigo mesmo por ler este livro. Você já está aprendendo mais do que a maioria das pessoas do setor de *software,* pois um livro é mais do que a maioria dos programadores leem a cada ano (DeMarco e Lister 1999). Uma pequena leitura faz muito pelo avanço profissional. Se você ler um bom livro sobre programação a cada dois meses, aproximadamente 35 páginas por semana, em breve terá um firme domínio da área e se distinguirá de praticamente todo mundo a sua volta.

Una-se a outros profissionais Encontre outras pessoas que se preocupam em aumentar suas habilidades de desenvolvimento de *software.* Participe de uma conferência, junte-se a um grupo de usuários local ou participe de um grupo de discussão *online.*

Comprometa-se com o desenvolvimento profissional Bons programadores procuram permanentemente maneiras de melhorar. Considere a seguinte escada de desenvolvimento profissional, usada em várias empresas:

- **Nível 1: Início** Iniciante é um programador capaz de usar os recursos básicos de uma linguagem. Essa pessoa pode escrever classes, rotinas, loops e condicionais, e usar vários recursos de uma linguagem.
- **Nível 2: Introdutório** Programador intermediário é aquele que já passou da fase de iniciante; é capaz de usar os recursos básicos de várias linguagens e se sente muito à vontade com pelo menos uma linguagem.
- **Nível 3: Competência** Um programador competente tem experiência em uma linguagem, em um ambiente ou em ambos. Um programador nesse nível pode saber todos os detalhes do J2EE ou ter o *Annotated C++ Reference Manual* (manual de referência comentado da linguagem C++) memorizado. Esta categoria de programadores é extremamente valiosa para suas empresas; a maioria nunca ultrapassa esse nível.
- **Nível 4: Liderança** Um líder tem a experiência de um programador de Nível 3 e reconhece que programar é apenas 15% de comunicação com o computador e 85% de comunicação com as pessoas. Apenas 30% do tempo de um programador normal é gasto trabalhando sozinho (McCue 1978). Menos tempo ainda é gasto na comunicação com o computador. O guru escreve código para uma audiência de pessoas e não de máquinas. Os programadores com verdadeiro nível de guru escrevem códigos claros como cristal e também o documentam. Eles não desperdiçam sua valiosa massa cinzenta reconstruindo a lógica de uma seção de código que poderiam ter lido em um comentário de uma frase.

Um codificador excelente que não dê ênfase à legibilidade provavelmente fica preso no Nível 3, mas, normalmente, nem mesmo isso acontece. De acordo com minha experiência, o principal motivo pelo qual as pessoas escrevem um código ilegível é que o código delas é ruim. Elas não dizem para si mesmas "Meu código é ruim; portanto, vou torná-lo difícil de ler". Elas simplesmente não entendem seu código

suficientemente bem a ponto de torná-lo legível, o que as prende em um dos níveis inferiores.

O pior código que já vi foi escrito por uma pessoa que não permitia que ninguém chegasse perto de seus programas. Finalmente, o gerente a ameaçou de demissão, caso não cooperasse. O código dessa pessoa não tinha comentários e era repleto de variáveis x, xx, xxx, xx1 e xx2 espalhadas, todas elas globais. O chefe do gerente imaginava que ela era uma excelente programadora, porque corrigia erros rapidamente. A qualidade do código dela dava-lhe muitas oportunidades para demonstrar sua capacidade de corrigir erros.

Não é nenhum pecado ser iniciante ou intermediário. Não é nenhum pecado ser um programador competente, em vez de líder. O pecado está em quanto tempo você permanece como iniciante ou intermediário, depois de saber o que tem de fazer para melhorar.

13.4 Honestidade intelectual

Parte do amadurecimento do profissional da programação é desenvolver um profundo senso de honestidade intelectual. A honestidade intelectual normalmente se manifesta de várias maneiras, a saber:

- Recusando-se a fingir que você é especialista, quando não é
- Admitindo prontamente seus erros
- Tentando entender um alerta do compilador, em vez de suprimir a mensagem
- Entendendo claramente seu programa - e não compilando-o para ver se funciona
- Fornecendo relatórios de status realistas
- Fornecendo estimativas de cronograma realistas e mantendo-se firme quando o gerente pedir para você ajustá-las

Os dois primeiros itens dessa lista - admitir que você não sabe algo ou que cometeu um erro - ecoam o tema da humildade intelectual. Como você pode aprender algo novo se finge que já sabe tudo? É melhor fingir que você não sabe nada. Ouça as explicações das pessoas, aprenda algo de novo com elas e avalie se elas sabem do que elas estão falando.

Esteja pronto para quantificar seu grau de certeza sobre qualquer assunto. Se ele normalmente for de 100%, esse é um sinal de alerta.

Recusar-se a admitir erros é um hábito particularmente irritante. Se Fulano se recusa a admitir um erro, aparentemente acredita que não admiti-lo fará com que os outros acreditem que ele não o cometeu. Na verdade, ocorre o oposto. Todo mundo saberá que ele cometeu um erro. Os erros são aceitos como parte do fluxo e refluxo das atividades intelectuais complexas e, desde que não tenha havido negligência, ninguém fará qualquer cobrança ou deboche.

Caso Fulano se recuse a admitir um erro, a única pessoa que enganará será ele mesmo. Os outros saberão que estão trabalhando com um programador arrogante, que não é completamente honesto. Esse é um defeito mais grave do que cometer um simples erro. Se você cometer um erro, admita isso rápida e enfaticamente.

Fazer de conta que entende as mensagens do compilador, quando você não entende, é outro equívoco comum. Se você não entende um alerta do compilador ou acredita que sabe o que ele significa, mas está pressionado demais pelo tempo para verificá-lo, adivinhe o que é realmente uma perda de tempo? Você provavelmente acabará tentando resolver o problema de cima a baixo, enquanto o compilador acena com a solução à sua frente. Várias pessoas já me pediram ajuda na depuração de programas. Eu pergunto se elas têm uma compilação limpa e elas dizem que sim. Então, elas começam a explicar os sintomas do problema e eu digo “Hummmm... Isso parece ser um ponteiro não iniciado,

mas o compilador deve ter alertado a respeito". Então, elas dizem "Ah, sim - ele alertou sobre isso. Mas pensamos que significava outra coisa". É difícil enganar outras pessoas sobre seus erros. É ainda mais difícil enganar o computador; portanto, não perca seu tempo tentando fazer isso.

Outro tipo de desleixo intelectual ocorre quando você não entende bem seu programa e "apenas o compila para ver se funciona". Um exemplo é executar o programa para ver se você deve usar < ou <=. Nessa situação, não importa realmente se o programa funciona, pois você não o entende bem o suficiente para saber por que ele funciona. Lembre-se que o teste consegue apenas mostrar a presença de erros e não sua ausência. Se você não entende o programa, não pode testá-lo completamente. Sentir-se tentado a compilar um programa para "ver o que acontece" é um sinal de alerta. Isso pode significar que você precisa voltar ao projeto ou que começou a codificar antes de ter certeza de que sabia o que estava fazendo. Certifique-se de ter um forte domínio intelectual sobre o programa, antes de abandoná-lo no compilador.

Os relatórios de status representam uma área de escandalosa duplicidade. Os programadores são notórios em dizer que um programa está "90% completo", durante os últimos 50% do projeto. Se o seu problema é ter uma ideia inadequada de seu próprio progresso, você pode resolvê-lo aprendendo mais sobre como trabalha. Mas se o seu problema é que você não fala francamente porque quer dar a resposta que seu gerente deseja ouvir, então a história é diferente. Normalmente, um gerente gosta de observações honestas sobre o status de um projeto, mesmo que o conteúdo delas não seja o que ele quer ouvir. Se as suas observações são produto da reflexão, exprima-as da forma mais imparcial que você puder e privativamente. A gerência precisa ter informações precisas para coordenar as atividades de desenvolvimento e uma cooperação total é essencial.

Um problema relacionado com os relatórios de status imprecisos é a estimativa imprecisa. O cenário típico é o seguinte: a gerência pede para Siclano fazer uma estimativa de quanto tempo levará para desenvolver um novo produto de banco de dados. Siclano fala com alguns programadores, calcula alguns números e volta com uma estimativa de oito programadores e seis meses. Seu gerente diz "Não é isso que estamos buscando. Você poderia fazer isso em um tempo mais curto, com menos programadores?" Siclano sai, pensa sobre o assunto e decide que por um curto espaço de tempo, poderia cortar o período de treinamento e de férias, e fazer todo mundo trabalhar algumas horas extras. Ele volta com uma estimativa de seis programadores e quatro meses. Seu gerente diz "Isso está ótimo. Esse é um projeto de prioridade relativamente baixa; portanto, tente mantê-lo dentro do cronograma sem quaisquer horas extras, pois o orçamento não permitiria isso".

O erro que Siclano cometeu foi não perceber que as estimativas não são negociáveis. Ele pode revisar uma estimativa para ser mais preciso, mas negociar com seu chefe não mudaria o tempo que leva para desenvolver um projeto de software. Bill Weimer, da IBM, afirma: "Descobrimos que o pessoal técnico, em geral, era muito bom para fazer estimativas de requisitos de projeto e cronogramas. O problema que eles tinham era defender suas decisões; eles precisavam aprender a não ceder terreno". Siclano não tornará seu gerente satisfeito, prometendo entregar um projeto em quatro meses e entregando-o em seis, mais do que o faria prometendo e entregando-o em seis meses. Ele perderia credibilidade pelo comprometimento e ganharia respeito mantendo-se firme em sua estimativa.

Se a gerência pressionar para mudar sua estimativa, perceba que, em última análise, a decisão de fazer um projeto cabe a ela: "Veja. O projeto vai custar isto. Não posso dizer se ele vale esse preço para a empresa - esse é o seu trabalho. Mas posso dizer quanto demora para desenvolver um software - esse é o meu trabalho. Não posso 'negociar' o tempo que ele exigirá; isso é como negociar quantos metros existem

em um quilômetro. Não se pode negociar as leis da natureza. Entretanto, podemos negociar outros aspectos do projeto que afetam a agenda e, então, estimá-la novamente. Podemos eliminar recursos, reduzir o desempenho, desenvolver o projeto em incrementos, ou usar menos pessoas e um cronograma mais longo, ou mais pessoas e um cronograma mais curto".

Uma das trocas mais alarmantes de que já ouvi falar foi em um discurso sobre gerenciamento de projetos de software. O orador era o autor de um livro campeão de vendas sobre gerenciamento de projetos de software. Um membro da audiência perguntou; "O que você faz se a gerência pede uma estimativa e você sabe que, se fornecer uma estimativa precisa, ela dirá que é alta demais e decidirá não fazer o projeto?". O orador respondeu que essa era uma daquelas áreas complicadas em que você tinha que fazer a gerência aceitar o projeto, estimando-o por baixo. Ele disse que, quando tivessem investido na primeira parte do projeto, o levariam a cabo até o fim.

Resposta errada! A gerência é responsável pelos problemas gerais do andamento da empresa. Se determinado recurso de software vale US$ 250.000 para uma empresa e você estima que ele custará US$ 750.000 para desenvolver, a empresa não deverá desenvolver esse software. É responsabilidade da gerência fazer tais julgamentos. Quando o orador defendeu a mentira sobre o custo do projeto, dizer à gerência que ele custaria menos do que realmente deveria custar, defendeu veladamente roubar a autoridade da gerência. Se você crê que um projeto é interessante, que ele abre importantes possibilidades para a empresa ou oferece treinamento valioso, diga isso claramente. A gerência também pode ponderar esses fatores. Mas enganar a gerência, fazendo-a tomar a decisão errada, poderia custar literalmente à empresa centenas de milhares de dólares. Se isso custar o seu emprego, você terá recebido o que merece.

13.5 Comunicação e cooperação

Programadores verdadeiramente excelentes aprendem a trabalhar e a se portar bem com os outros. Escrever um código legível é inerente ao fato de integrar uma equipe. O computador provavelmente lê seu programa com a mesma frequência que outras pessoas, mas ele está mais bem preparado para ler um código malfeito do que as pessoas.

Como diretriz de legibilidade, lembre-se da pessoa que terá de modificar seu código. Programar é primeiro comunicar-se com outro programador e depois comunicar-se com o computador.

13.6 Criatividade e disciplina

> *Quando deixei a faculdade, eu acreditava que era o melhor programador do mundo. Eu podia escrever um programa de jogo-da-velha imbatível, usar cinco linguagens de computador diferentes e criar programas de 1.000 linhas que FUNCIONAVAM (realmente!). Então, caí no Mundo Real. Minha primeira tarefa no Mundo Real foi ler e entender um programa de 200.000 linhas em Fortran e depois torná-lo duas vezes mais rápido. Qualquer Programador de Verdade lhe dirá que toda Codificação Estruturada do mundo não ajudaria a resolver um problema como esse - isso exige talento de verdade. - EdPost*

É difícil explicar para um recém-graduado em ciência da computação por que você precisa de convenções e disciplina de engenharia. Quando eu era estudante, o maior programa que escrevi tinha cerca de 500 linhas de código executável. Como profissional, já escrevi dezenas de utilitários menores do que 500 linhas, mas o tamanho médio de projetos importantes tem sido de 5.000 a 25.000 linhas, e tenho participado de projetos com mais de meio milhão de linhas de código. Esse tipo de

trabalho não exige as mesmas habilidades em uma escala maior; mas um conjunto de habilidades completamente novo.

Alguns programadores criativos veem a disciplina dos padrões e convenções como algo sufocante para sua criatividade. A verdade, no entanto, é o oposto. Você consegue imaginar um site da Web no qual cada página usasse diferentes fontes, cores, alinhamento de texto, estilos gráficos e dicas de navegação? O efeito seria caótico, não criativo. Sem padrões e convenções nos grandes projetos, sua conclusão é impossível. A criatividade nem mesmo é imaginável. Não desperdice sua criatividade em coisas que não têm importância. Estabeleça convenções nas áreas que não são críticas, para que você possa concentrar suas energias criativas no que realmente interessa.

Em uma retrospectiva de 15 anos de trabalho no Laboratório de Engenharia de Software da NASA, McGarry e Pajerski relataram que os métodos e ferramentas que dão ênfase à disciplina humana têm sido especialmente eficientes. Muitas pessoas dotadas de alta criatividade têm sido extremamente disciplinadas. "A formalidade liberta", como diz o ditado. Grandes arquitetos trabalham dentro das restrições dos materiais físicos, do tempo e do custo. Os grandes artistas, também. Qualquer um que tenha examinado os desenhos de Leonardo da Vinci tem de admirar sua atenção disciplinada ao detalhe. Quando Michelangelo projetou o teto da Capela Sistina, ele o dividiu em conjuntos simétricos de formas geométricas, como triângulos, círculos e quadrados. Ele a projetou em três zonas, correspondendo aos três estágios platônicos. Sem essa estrutura e disciplina imposta a si mesmo, as 300 figuras humanas teriam sido apenas caóticas, em vez dos elementos coerentes de uma obra-prima artística.

Uma obra-prima da programação exige a mesma disciplina. Se você não tentar analisar os requisitos e o projeto antes de começar a codificar, grande parte de seu aprendizado sobre o projeto ocorrerá durante a codificação e o resultado de seu trabalho parecerá mais uma pintura com os dedos, feita por uma criança de três anos de idade, do que uma obra de arte.

13.7 Preguiça

A preguiça se manifesta de várias maneiras:

- No adiamento de uma tarefa desagradável
- No realizar uma tarefa desagradável rapidamente, para tirá-la do caminho
- No escrever uma ferramenta para realizar a tarefa desagradável, para que nunca mais tenha que fazê-la

Algumas dessas manifestações da preguiça são melhores do que outras. No primeiro caso, dificilmente poderá ser benéfica. Você provavelmente já passou pela experiência de perder várias horas fazendo trabalhos que na verdade não precisavam ser feitos, para não ter que encarar uma tarefa relativamente menor que não podia evitar. Eu detesto entrada de dados e muitos programas exigem um pequeno volume de entrada de dados. Já retardei o trabalho em um programa por vários dias, apenas para adiar a inevitável tarefa de inserir várias páginas de números manualmente. Esse hábito é "preguiça pura". Ela se manifesta novamente no hábito de compilar uma classe para ver se ela funciona e, com isso, evitar o exercício de conferir a classe mentalmente.

As pequenas tarefas nunca são tão más quanto parecem. Se você desenvolver o hábito de resolvê-las imediatamente, poderá evitar o tipo de preguiça da protelação. Esse hábito é a "preguiça esclarecida" - o segundo tipo de preguiça. Você ainda sente preguiça, mas está contornando o problema passando a menor quantidade de tempo possível em algo que é desagradável.

O terceiro caso é escrever uma ferramenta para realizar a tarefa desagradável.

Essa é a "preguiça a longo prazo". Sem dúvida, esse é o tipo mais produtivo de preguiça (desde que você economize mesmo tempo, após ter escrito a ferramenta). Nesses contextos, uma certa quantidade de preguiça é benéfica.

Quando você fica diante do espelho, vê o outro lado da fotografia da preguiça. "Empurrão" ou "fazer um esforço" não tem a mesma graça que tinha na aula de educação física do colégio. Empurrão é esforço extra e desnecessário. Ele mostra que você está ansioso, mas não que irá fazer seu trabalho. É fácil confundir movimento com progresso, atividade com produtividade. O trabalho mais importante na programação eficiente é o pensamento, e as pessoas tendem a não parecer ocupadas quando estão pensando. Se eu trabalhasse com um programador que parecesse ocupado o tempo todo, iria presumir que ele não era um bom programador, pois não estaria usando sua ferramenta mais valiosa: o cérebro.

13.8 Características que não importam tanto quanto você poderia pensar

O empurrão não é a única característica que você poderia admirar em outros aspectos de sua vida, mas isso não funciona muito bem no desenvolvimento de software.

Persistência

Dependendo da situação, a persistência pode ser uma vantagem ou um risco. Assim como a maioria dos conceitos de valor, ela é identificada por diferentes palavras, dependendo de você qualificar se é boa ou ruim. Se você quer identificar a persistência como uma particularidade negativa, trate-a como "teimosia" ou "obstinação". Se quer identificá-la como uma qualidade, trata-a como "tenacidade" ou "perseverança".

Na maioria das vezes, a persistência no desenvolvimento de software é mesmo obstinação - ela tem pouco valor. Quando você está travado no trecho de um novo código, dificilmente a persistência é uma virtude. Tente refazer o projeto da classe, tente uma estratégia de codificação alternativa ou tente voltar a ela posteriormente. Quando uma estratégia não está funcionando, esse é um bom momento para buscar outra alternativa.

Na depuração, pode ser extremamente satisfatório rastrear o erro que vem lhe aborrecendo há quatro horas, mas em geral é melhor desistir do erro após algum tempo sem nenhum progresso - digamos, 15 minutos. Permita que seu subconsciente reflita sobre o problema por alguns instantes. Tente pensar em uma estratégia alternativa que contornaria completamente o problema. Reescreva desde o início a seção de código problemática. Volte a ela posteriormente, quando sua mente estiver descansada. Lutar com problemas de computador não é uma virtude. Evitá-los é melhor.

É difícil saber quando abandonar, mas é essencial que você se pergunte quando. Ao perceber que está frustrado, esse é um bom momento para fazer a pergunta. Perguntar não significa necessariamente que é hora de abandonar, mas provavelmente significa que é hora de definir alguns parâmetros para a atividade: "Se eu não resolver o problema usando esta estratégia dentro dos próximos 30 minutos, vou me conceder mais 10 minutos para ter uma ideia genial sobre alternativas diferentes e tentarei a melhor na próxima hora".

Experiência

O valor da experiência prática, quando comparada ao aprendizado com livros, é menor no desenvolvimento de software do que em muitos outros campos, por vários motivos. Em muitos outros campos, o conhecimento básico muda num ritmo lento, suficiente para que alguém que se formou na faculdade 10 anos depois de você tenha

recebido basicamente o mesmo conhecimento sobre as disciplinas. No desenvolvimento de software, entretanto, até o conhecimento básico muda rapidamente. A pessoa que se formou na faculdade 10 anos depois de você provavelmente aprendeu duas vezes mais sobre técnicas de programação eficientes. Os programadores mais antigos tendem a ser vistos com suspeita, não apenas porque poderiam não estar a par da tecnologia específica, mas porque podem nunca ter visto os conceitos de programação básicos que se tornaram conhecidos depois que deixaram a universidade.

Em outros campos, o que você aprende hoje sobre seu trabalho provavelmente o ajudará em sua atividade amanhã. Já em termos de software, se você não puder livrar-se dos vícios de pensamento que desenvolveu enquanto usava sua primeira linguagem de programação ou das técnicas de otimização de código que funcionavam em sua antiga máquina, sua experiência será pior do que não ter nenhuma. Muitos profissionais de software passam seu tempo se preparando para lutar a última batalha, em vez da próxima. Quando você não consegue adaptar-se aos novos tempos, a experiência é mais um obstáculo do que uma ajuda.

Não bastassem as rápidas mudanças no desenvolvimento de software, ocorre ainda que as pessoas frequentemente tiram conclusões erradas a partir de suas experiências. É difícil ver sua própria vida objetivamente. Você pode ignorar elementos importantes de sua experiência, que o fariam tirar conclusões diferentes se os reconhecesse. Dedicar-se a estudos de outros programadores é muito útil, pois eles revelam a experiência de outras pessoas - filtrada o suficiente para que você possa examiná-la objetivamente.

As pessoas também dão uma ênfase absurda à quantidade de experiência que os programadores têm. “Queremos um programador com cinco anos de experiência de programação em C” é uma exigência tola. Se um programador não tiver aprendido C após um ou dois anos, os próximos três anos não farão muita diferença. Esse tipo de “experiência” tem pouca relação com o desempenho.

O fato de a informação mudar rapidamente na programação cria dinâmicas estranhas na área da “experiência”. Em muitos campos, um profissional que tenha uma história de realizações pode parar, relaxar e usufruir o respeito conquistado por uma carreira de sucessos. No desenvolvimento de software, qualquer um que pare se torna rapidamente desatualizado. Para permanecer valioso, você precisa estar atualizado. Para programadores jovens e ávidos, isso é uma vantagem. Os programadores mais antigos às vezes sentem que já obtiveram seu galardão e se ressentem de ter de provar a si mesmos ano após ano.

A questão sobre a experiência é esta: se você trabalha há 10 anos, tem 10 anos de experiência ou 1 ano de experiência 10 vezes? Você precisa refletir sobre suas atividades para obter uma experiência verdadeira. Se você fizer do aprendizado um comprometimento contínuo, terá experiência. Caso contrário, permanecerá inexperiente, não importa quantos anos de janela tenha.

Programação abusiva

> *Se você não tiver passado pelo menos um mês trabalhando no mesmo programa - dedicando-se a ele 16 horas por dia, sonhando com ele durante as 8 horas restantes de sono intranquilo, virando várias noites, tentando eliminar aquele "último erro" do programa –, então não escreveu um programa de computador realmente complexo. E você pode não ter ideia de que há algo divertido na programação. - Edward Yourdon*

Esse vigoroso tributo à virilidade na programação é pura besteira e uma receita quase certa para falhas. Aquelas tarefas de programação que duram a noite toda fazem você se sentir o maior programador do mundo; depois, é preciso passar várias semanas

corrigindo os defeitos instalados durante seu momento de glória. Definitivamente, entusiasme-se com a programação. Mas saiba que o entusiasmo não substitui a competência. Priorize o que é mais importante.

13.9 Hábitos

> *As virtudes morais, então, são produzidas em nós não pela natureza nem contra ela....sua total manifestação em nós se dá devido ao hábito.... Tudo que temos que aprender, aprendemos fazendo.... Os homens se tornarão bons construtores como resultado de construírem bem, e maus construtores como resultado de construírem mal....Portanto, não têm pouca importância os tipos de hábitos que formamos na primeira idade - eles fazem uma diferença enorme, ou ainda, fazem toda a diferença no mundo.- Aristóteles*

Bons hábitos são importantes, porque a maior parte do que você faz como programador o faz sem pensar conscientemente a respeito. Por exemplo, tempos atrás, você pode ter pensado a respeito de como queria formatar loops indentados, mas agora pensa sobre isso novamente sempre que escreve um novo loop. Por hábito, você faz isso da maneira que faz. Isso vale para praticamente todos os aspectos da formatação de programas. Quando foi a última vez que você questionou seriamente seu estilo de formatação? São boas as chances de que, se você vem programando há cinco anos, tenha questionado isso pela última vez há quatro anos e meio. No resto de tempo, você contou com o hábito.

Você tem hábitos em muitas áreas. Por exemplo, os programadores tendem a verificar os índices de loop cuidadosamente e a não verificar as instruções de atribuição, o que torna os erros em instruções de atribuição muito mais difíceis de encontrar do que os erros em índices de loop. Você responde às críticas de uma maneira amistosa ou de uma maneira hostil. Você está sempre procurando maneiras de tornar o código legível ou rápido ou não está. Se você tem de escolher entre tornar o código rápido ou torná-lo legível e sempre faz a mesma escolha, não está realmente escolhendo - está respondendo ao hábito.

Estude a citação de Aristóteles e substitua “virtudes morais” por “virtudes da programação”. Ele destaca que você não está predisposto a um bom ou mau comportamento, mas é constituído de tal maneira que pode se tornar um bom ou mau programador. A principal maneira para se tornar bom ou mau no que você faz é fazendo - os construtores, construindo; os programadores, programando. O que você faz se torna hábito e, com o passar do tempo, seus hábitos bons e maus determinam se você é bom ou mau programador.

Bill Gates afirma que todo programador que deverá ser bom, já o é nos primeiros anos. Depois disso, se um programador é bom ou não, é o mesmo que esculpir na água. Depois de você estar programando há muito tempo, é difícil começar repentinamente a dizer: “Como torno este loop mais rápido?” ou “Como torno este código mais legível?”. Esses são hábitos que os bons programadores desenvolvem desde cedo.

Quando você for aprender algo pela primeira vez, aprenda da maneira certa. Quando fizer pela primeira vez, você estará pensando ativamente sobre isso e ainda terá uma escolha fácil entre fazer de maneira correta e fazer de maneira errada. Após fazer alguma coisa várias vezes, você presta menos atenção ao que está fazendo e a “força do hábito” entra em ação. Certifique-se de que os hábitos que entrem em ação sejam aqueles que você deseja ter.

E se você ainda não for portador dos hábitos mais eficazes? Como você muda um mau hábito? Se eu tivesse a resposta definitiva, poderia vender programas de auto-ajuda de madrugada na TV. Mas aqui está pelo menos parte de uma resposta. Você não pode substituir um mau hábito por nenhum hábito. É por isso que as pessoas que

param de fumar, param de dizer palavrões ou param de comer demais repentinamente passam por maus momentos, a menos que substituam isso por outra coisa, como mascar chicletes. É mais fácil substituir um hábito antigo por um novo do que eliminá-lo completamente. Na programação, tente desenvolver novos hábitos que funcionem. Desenvolva os hábitos de escrever uma classe em pseudocódigo antes de codificá-la, e de ler cuidadosamente o código antes de compilá-lo, por exemplo. Você não terá de se preocupar com a perda dos maus hábitos; eles serão naturalmente deixados de lado, quando os novos hábitos tomarem seus lugares.

Pontos-chave

- Seu caráter pessoal afeta diretamente sua capacidade de escrever programas de computador.
- As características que mais importam são a humildade, a curiosidade, a honestidade intelectual, a criatividade, a disciplina e a preguiça esclarecida.
- As características de um programador superior quase nada têm a ver com talento, mas têm tudo a ver com um comprometimento com o desenvolvimento pessoal.
- Surpreendentemente, a inteligência bruta, a experiência, a persistência e a coragem tanto prejudicam como ajudam.
- Muitos programadores não buscam novas técnicas e informações ativamente; em vez disso, contam com uma exposição acidental, no trabalho, às novas informações. Se você dedicar um pequeno percentual de seu tempo à leitura e ao aprendizado sobre programação, após alguns meses ou anos se distinguirá significativamente da corrente principal da programação.
- Um bom caráter é principalmente uma questão de ter os hábitos corretos. Para ser um ótimo programador, desenvolva bons hábitos e o resto virá naturalmente.