Filosofia Unix

Origem: Wikipédia, a enciclopédia livre.

A **filosofia Unix** é um conjunto de normas culturais e abordagens filosóficas para o desenvolvimento de *software*, criada com base na experiência de alguns dos principais desenvolvedores dos sistemas operacionais da família Unix.

Índice

McIlroy: Um quarto de século de Unix

Doug McIlroy, o inventor da canalização e um dos fundadores da tradição do Unix, resumiu a filosofia da seguinte maneira[1]:

> "Esta é a filosofia Unix:
>
> > Escreva programas que façam apenas uma coisa mas que façam bem feito.
> > Escreva programas que trabalhem juntos.
> > Escreva programas que manipulem *streams* de texto, pois esta é uma interface universal."

Ou, de maneira simples, como: "faça apenas uma coisa e faça bem".

Dos três princípios, apenas o terceiro é específico do Unix, embora os desenvolvedores Unix enfatizem todos os três princípios mais freqüentemente que outros desenvolvedores.

Pike: Notas para programação na linguagem C

Rob Pike ofereceu as seguintes "regras" em *Notas para programação em C* (http://www.lysator.liu.se/c/pikestyle.html) como máximas da programação de computadores, embora elas possam ser vistas como pontos da filosofia Unix:

- Regra 1: Você não pode dizer qual trecho de um programa é lento. Os gargalos ocorrem em locais que surpreendem, portanto, não tente supor e determinar uma solução até que tenha determinado exatamente onde se encontra o gargalo.
- Regra 2: Meça. Não otimize o programa até que você tenha medido o seu tempo de execução, e mesmo depois de medido o tempo, não otimize a menos que uma parte do código esteja gastando muito mais tempo em comparação com o restante do programa.
- Regra 3: Algoritmos extravagantes são lentos quando n é pequeno, e n é normalmente pequeno. Algoritmos extravagantes têm grandes constantes. Até que você saiba que n torna-se freqüentemente grande, não seja extravagante. (Mesmo se n tornar-se grande, use a Regra 2 primeiro.)
- Regra 4: Algoritmos extravagantes contém mais defeitos que algoritmos simples e são mais difíceis de implementar. Utilize algoritmos simples assim como estrutura de dados simples.

- Regra 5: O dado domina. Se você escolher a estrutura de dados certa e organizar bem as coisas, os algoritmos surgirão naturalmente. O elemento central da programação é a estrutura de dados, não o algoritmo.
- Regra 6: Não existe Regra 6.

As regras 1 e 2 de Pike, reforçam a famosa máxima de Tony Hoare: "A otimização prematura é a raiz de todo mal." Ken Thompson reescreveu as regras 3 e 4 da seguinte forma: "Quando em dúvida, use força bruta." As regras 3 e 4 são exemplos da filosofia de projeto KISS. A regra 5 foi colocada anteriormente por Fred Brooks no livro *The Mythical Man-Month*. A regra 5 é freqüentemente resumida como: "escreva um código burro que use uma estrutura de dados inteligente", e é um exemplo da norma "Se a sua estrutura de dados é boa o bastante, o algoritmo para manipulá-la deverá ser trivial."

Mike Gancarz: The UNIX Philosophy

Em 1994, Mike Gancarz, membro da equipe que projetou o X Window System, combinou a sua experiência com discussões que teve com programadores e pessoas que dependem do Unix para produzir o livro *The UNIX Philosophy*, que resume os pontos citados acima em 9 princípios supremos:

1. *O pequeno é belo.*
2. *Faça cada programa fazer uma coisa bem feita.*
3. *Faça um protótipo assim que possível.*
4. *Escolha "portabilidade" ao invés de eficiência.*
5. *Armazene dados em arquivos no formato texto.*
6. *Tire vantagem das funcionalidades do software.*
7. *Utilize scripts para incrementar a funcionalidade e portabilidade.*
8. *Evite construir interfaces do usuário que sejam engessadas.*
9. *Faça de cada programa um filtro.*

Os 10 princípios "inferiores" não são aceitos universalmente como parte da filosofia Unix e, em alguns casos, são tema de debates acalorados (com a discussão núcleo monolítico x micronúcleo):

1. *Permita ao usuário definir o ambiente.*
2. *Faça o núcleo do sistema operacional pequeno e leve.*
3. *Utilize letras minúsculas e faça textos curtos.*
4. *Salve as árvores.*
5. *O silêncio é ouro.*
6. *Pense paralelamente.*
7. *A soma de todas as partes é maior que o todo.*
8. *Preste atenção na lei do 90% da solução.*
9. *O pior é melhor.*
10. *Pense de forma hierárquica.*

O pior é melhor

Richard P. Gabriel sugere que a vantagem chave do Unix está relacionada a uma filosofia de projeto que ele chamou de "o pior é melhor" (*worse is better*). No estilo de projeto "pior é melhor", a simplicidade da interface "e" da implementação é mais importante que qualquer outro atributo do sistema — incluindo correção, consistência e inteireza. Gabriel sustenta que este estilo de projeto contém vantagens evolucionárias chave, embora ele questione a qualidade de alguns resultados.[2]

Raymond: A arte da programação Unix

Eric S. Raymond, no seu livro *The Art of Unix Programming*, resume a filosofia Unix como sendo a filosofia KISS. Ele descreve a sua crença de como esta filosofia é aplicada como uma norma cultural do Unix, apesar de, previsivelmente, não ser difícil encontrar violações severas nas práticas adotadas em projetos Unix atuais [1]:

- Regra da "modularidade": escreva partes simples conectadas por interfaces simples.
- Regra da clareza: clareza é melhor que inteligência.
- Regra da composição: projete os programas para serem conectados com outros programas.
- Regra da separação: separe a configuração dos mecanismos; separe as interfaces da implementação de algoritmos.
- Regra da simplicidade: projete para simplicidade; adicione complexidade apenas onde é necessário.

- Regra da parcimônia: escreva um programa de tamanho grande somente quando fica claro, por demonstração, que de outra maneira não seria possível resolver o problema.
- Regra da transparência: projete de forma a dar visibilidade, para fazer do processo de depuração e inspeção uma tarefa fácil.
- Regra da robustez: a robustez é a filha da transparência e simplicidade.
- Regra da representação: armazene o conhecimento na forma de dados, para que a lógica dos programas possa ser robusta e estúpida.
- Regra da menor surpresa: no projeto de interfaces, sempre faça aquilo que é o menos surpreendente.
- Regra do silêncio: quando um programa não tem nada surpreendente para dizer, ele não deve dizer nada.
- Regra do reparo: quando é inevitável falhar, falhe ruidosamente e o mais cedo possível.
- Regra da economia: o tempo do programador é caro; conserve este tempo em detrimento do tempo da máquina.
- Regra da geração de código: evite escrever código; escreva programas que gerem código quando é possível.
- Regra da otimização: faça protótipo antes do refinamento. Coloque em funcionamento antes de ser otimizado.
- Regra da diversidade: desconfie de todas as alegações sobre a existência de um "modo correto de fazer as coisas".
- Regra da extensibilidade: projete para o futuro, pois ele chegará antes do que você imagina.

Muitas destas normas foram aceitas fora da comunidade Unix, mesmo que isso não tenha ocorrido num primeiro momento. Também é preciso dizer que muitas delas não foram criadas pela comunidade Unix. Todavia, os adeptos da programação Unix tendem a aceitar uma combinação destas idéias como a fundação do estilo Unix.

Controvérsia

Existe uma controvérsia sobre até onde os trabalhos do projeto GNU, da Free Software Foundation, seguem os princípios da filosofia do Unix. Muitos poderiam dizer que não, pois as ferramentas GNU são maiores e contém um número maior de funcionalidades, em comparação com as suas equivalentes do Unix.

Já em 1983, Rob Pike escreveu um artigo criticando a expansão de funcionalidades básicas do Unix, como no caso do comando cat, implementada nos sistemas BSD.[3] Esta questão somente se tornou relevante com o surgimento do projeto GNU e de outras variações comerciais do Unix, que concentraram um grande número de funcionalidades em alguns programas.

Frases

- "*Unix is simple. It just takes a genius to understand its simplicity.*" – Dennis Ritchie
- "*UNIX was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things.*" – Doug Gwyn
- "*Unix never says 'please'.*" – Rob Pike
- "*The UNIX philosophy basically involves giving you enough rope to hang yourself. And then a couple of feet more, just to be sure.*" - autoria desconhecida

Ver também

- Plan 9
- Canalização
- The Elements of Style – Uma das fontes de inspiração para a filosofia Unix.
- Engenharia de software

Referências

1. Eric S. Raymond. «The Art of Unix Programming» (http://www.faqs.org/docs/artu/index.html) (em inglês). Consultado em 13 de abril de 2007
2. Richard P. Gabriel. «Worse Is Better» (http://www.dreamsongs.com/WorseIsBetter.html) (em inglês). Consultado em 13 de julho de 2007
3. Rob Pike. «UNIX Style, or cat -v Considered Harmful» (http://gaul.org/files/cat_-v_considered_harmful.html) (em inglês). Consultado em 13 de abril de 2007

Bibliografia

- *A Quarter Century of Unix*, Peter H. Salus, Addison-Wesley, 31 de maio de 1994 (ISBN 0201547775)
- *The Art of Unix Programming*, Eric S. Raymond, Addison-Wesley, 17 de setembro de 2003 (ISBN 0131429019)
- *The UNIX Philosophy*, Mike Gancarz, Digital Press, 14 de dezembro de 1994 (ISBN 1555581234)

Ligações externas

- (em inglês) *The Art of Unix Programming* (http://www.catb.org/~esr/writings/taoup), Eric S. Raymond, Addison-Wesley, 17 de setembro de 2003 (ISBN 0131429019)
- (em inglês) The Unix Programming Environment (http://cm.bell-labs.com/cm/cs/upe/) por Brian Kernighan e Rob Pike, 1984
- (em inglês) *Notes on Programming in C* (http://www.lysator.liu.se/c/pikestyle.html), Rob Pike, 21 de setembro de 1989
- (em inglês) Joel on Software - Biculturalism (http://www.joelonsoftware.com/articles/Biculturalism.html)
