Introdução ao debugging de software

Date : 10 de Março de 2015

Por Luís Soares para o PPLWARE.COM

O *debugging* de um programa baseia-se em alguns princípios e técnicas transversais à maioria das linguagens e ambientes de programação. Tentarei, neste artigo, sintetizar o que entendo por *debugging*, introduzindo o tema, colocando algumas luzes nos conceitos fundamentais e mostrando que há um mundo para além de *alerts* e *prints*.

Debugging (ou depuração) é o processo pelo qual se identificam e corrigem *bugs* de *software* (ou *hardware*). Dominar o *debugging* é vital para um programador: os *bugs* vão inevitavelmente aparecer quando a complexidade (número de programadores, de requisitos, de linhas de código, de dependências, etc.) aumentar.
No início, fazer *debug* costuma ser algo chato e/ou custoso (pelo menos para mim, era); mas, com a prática, torna-se mais interessante, pois:

- faz-nos compreender melhor o *workflow* do programa, o que se passa “por baixo”, assim como tecnologias envolvidas;
- obriga a criar bom código que evita *bugs* semelhantes no futuro;
- alguns *bugs* estimulam a nossa capacidade de resolução de problemas e motivam-nos quando resolvidos.

Evitando o *debug*

A melhor forma de fazer debug é evitar ter de o fazer. Devemos **adotar técnicas e boas**

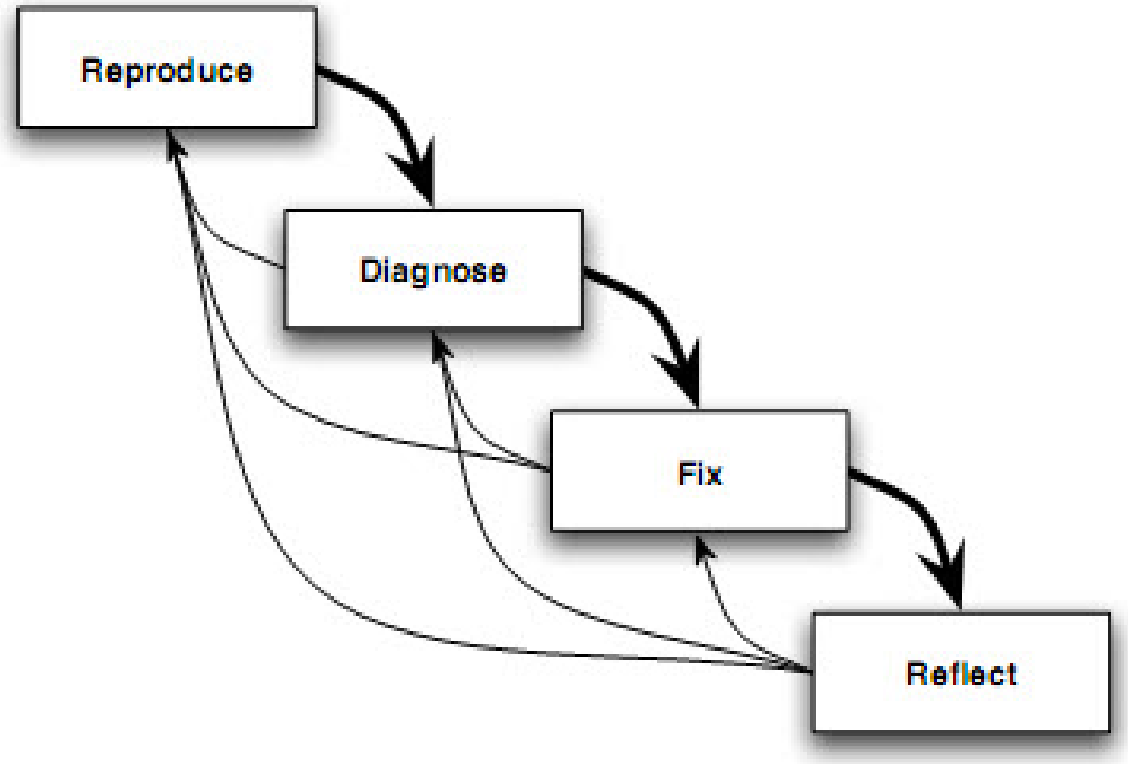
práticas que reduzam a probabilidade de ter bugs e outras que facilitem a sua correção caso ocorram. Eis algumas (cuja explicação sai fora do âmbito atual):

- *Code reviewing* e *Pair programming*;
- *Decoupling*, encapsulamento e modularidade;
- Uso correto de design patterns;
- TDD;
- Redução de dependências (ex. de tecnologias, de pessoas, do S.O.);
- Uso de c*onvention over configuration*;
- Princípios SOLID;
- Não faça *copy/paste* de código (nunca vale a pena);
- Mantenha o código limpo e tenha pouco código;
- Faça bons comentários;
- entre muitas outras boas práticas...

O processo de *debug*

O *debugging*, apesar de muito ligado à programação, é uma disciplina com o seu próprio processo. Mesmo não se pensando nisso, está-se implicitamente a segui-lo:

1. ***Reprodução***: saber os passos a seguir, as condições iniciais, assunções, etc;
2. ***Diagnóstico***: gravidade, prioridade, impactos, riscos, a zona em causa. Para este último:
3. ***Correção***;
4. ***Reflexão***: aplicação de medidas que garantam que o problema não se repete noutro formato: testes, documentação, validações de *input* e corner cases, criação de código mais resiliente, *refactoring*, etc.



Logo na reprodução do problema é frequente que o programador perceba o que está mal, passando logo para a correção. Se isso não suceder, passa-se então ao diagnóstico (o *debugging* propriamente dito), altura em que são aplicadas as seguintes técnicas (entre muitas outras):

- Fazer *reverse engineering*: análise de outputs e *stack traces*, de *dump files*, de pacotes de rede, de padrões de comportamento, entre outros;
- **Tentativa/erro**: quando não se tem a certeza dos contornos do problema e se colocam diversas hipóteses, seguindo-se a sua eliminação progressiva;
- **Isolar o problema**: por exemplo, comentando código não relevante no problema ou criando um *sample project* com o problema na sua versão mais simples;
- **Reverter o código** gradualmente, até se perceber o que causou o problema;
- ***Tracing (prints)***: a técnica mais comum: imprimir que se passou em algum lado e/ou variáveis disponíveis momentos;
- **Uso de** *debugger*: ferramenta por excelência para *debugging* (inclui controlo de fluxo, avaliador de expressões, consola, *profiler*, entre outros utilitários).

Vejamos o *tracing* e o **debugging** com ferramenta em mais detalhe.

Imprimindo (*tracing*)

Quem nunca fez *debug* com *prints*? *Tracing* é o método mais usado para *debug*. Fazer *tracing* é usar a consola (há quem use a própria GUI, embora isso não seja muito elegante...) para imprimir (i.e. exibir) o estado do programa sem bloquear o seu fluxo. Por outras palavras,

permite saber o valor de variáveis (ou simplesmente dizer que se passou lá) quando se passa nos sítios com *prints*. Pode ser usado para *debugging* ou apenas registo (*logging*). Eis alguns exemplos de *prints* em diferentes linguagens:

C	printf("Olá\n")
C++	