UFV
Universidade Federal de Viçosa

INF 112 - Programação II

Revisão de código e estratégias de depuração

Introdução

- Sistemas grandes, complexos, é possível garantir...
  - Código legível? Sem duplicidade?
  - Facilidade de manutenção?
  - Ausência de erros?
- Taxa média de detecção de defeitos
  - Testes unitários: 25%
  - Testes de integração: 45%
- Conseguimos melhorar esses valores?

Introdução

- Técnicas para "garantir" (e/ou melhorar) a qualidade do software desenvolvido
- Verificação
  - Software de acordo com a especificação
  - "Construímos o produto corretamente?"
- Validação
  - Software faz o que o usuário realmente deseja
  - "Construímos o produto certo (esperado)?"

Revisão de Código

- Tarefa construtiva de rever o código e a documentação para identificar erros de interpretação, incoerências e outras falhas
  - Confirmação externa (antes de alterações e inserções de novos códigos)
- Propósito:
  - Melhorar o código
  - Melhorar o programador

Revisão de Código

Benefícios

- Taxa média de detecção de defeitos
  - Inspeções de design e código: 55% - 60%
- 11 programas desenvolvidos (mesma equipe):
  - 5 (sem revisões): 4,50 erros a cada 100 LoC
  - 6 (com revisões): 0,82 erros a cada 100 LoC
- Conhecimento:
  - Melhor entendimento do código
  - Feedback/Programadores Iniciantes

FREEDMAN. D. P., and WEINBERG G. M. (1982) Software Inspections: An Effective Verification Process. IEEE Software.

Revisão de Código

Quem

- Desenvolvedor do código e o responsável pela revisão (desenvolvedor mais experiente), às vezes juntos pessoalmente, às vezes separados

Como

- Revisor dá sugestões de melhoria em um nível lógico e/ou estrutural, de acordo com conjunto previamente acordado de padrões de qualidade
- Correções são feitas até uma eventual aprovação do

Quando

- Após o autor de código finalizar uma alteração do sistema (não muito grande/pequena), que está pronta para ser incorporada ao restante

Revisão de Código @ Google

Modern Code Review: A Case Study at Google (https://ai.google/research/pubs/pub47025)

"All code that gets submitted needs to be reviewed by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian [Rietveld] to do code reviews, and obviously, we spend a good chunk of our time reviewing code."

– Amanda Camp, Software Engineer, Google

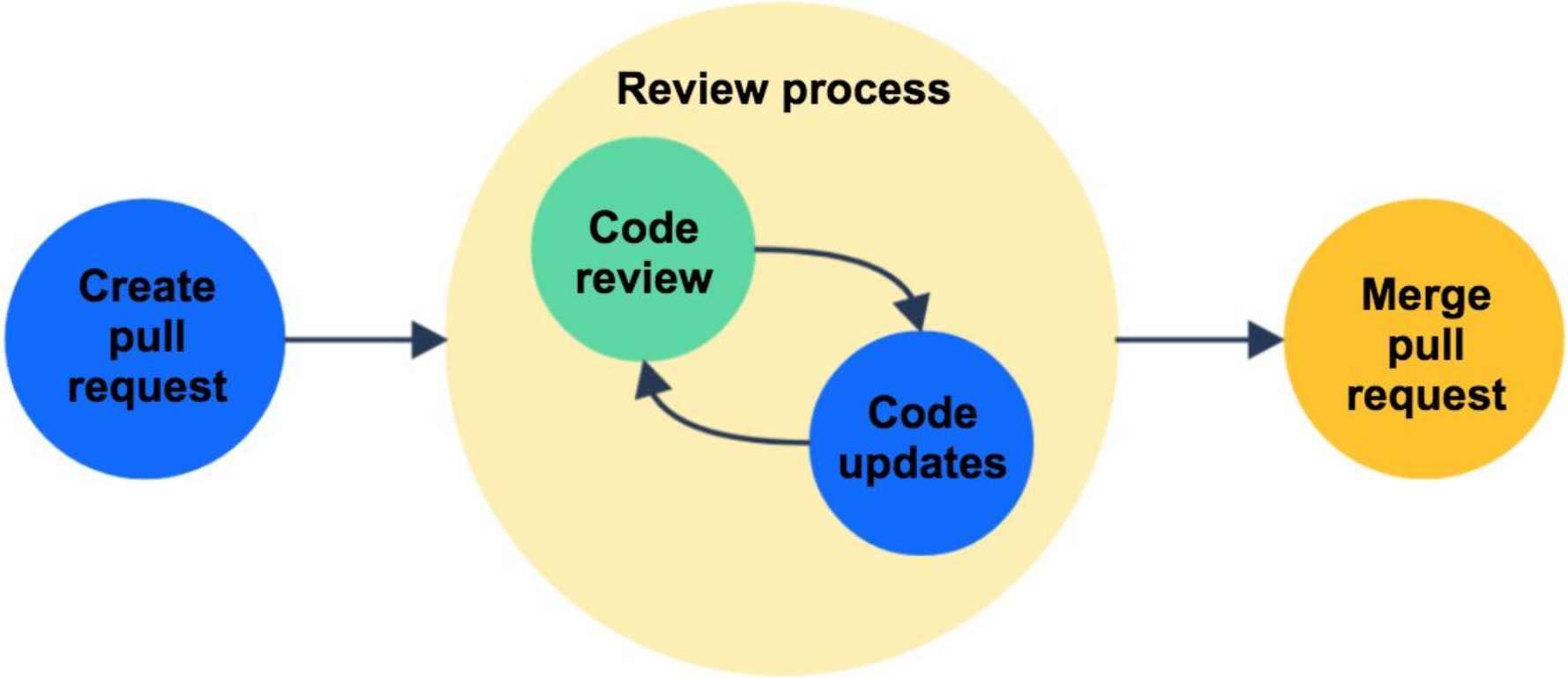
Tipos de Revisão

- **Email**
  - Olá, olhe meu código.
- **Ferramentas**
  - Gerrit
  - Rietveld
- **Ciclo de pull requests**
  - Github

Revisão de Código

No Github

- **Pull request**
  - Requisição para que o novo código seja integrado ao restante do sistema



Revisão de Código

Checklist

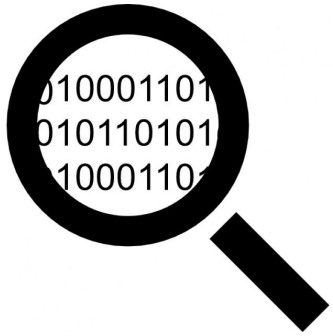
Revisão de código

Boas Práticas

- Estabeleça metas quantificáveis para revisão
- Utilize listas de verificação (checklists)
- Registre e verifique se os defeitos são consertados
- Você não irá revisar tudo de uma vez!
  - Talvez será adequado priorizar partes críticas

Depuração

- Verificação e validação
  - Relacionados ao estabelecimento da existência de falhas (inconsistências) em um programa
- Depuração (debugging)
  - Localização e reparação das falhas



Depuração
Motivação

- Depurar programas grandes é difícil (arte)
- Um bom programador deve:
  - conhecer uma ampla variedade de estratégias de depuração
  - conhecer/usar ferramentas que facilitam a depuração
    - Debuggers
    - Sistemas de controle de versão

Depuração vs. Testes

- Testar
  - O que fazer para tentar quebrar o programa?
- Depurar
  - O que fazer para tentar consertar o programa?

> *When debugging, novices insert corrective code; experts remove defective code.*
>
> *- R. Pattis*

Depuração

Tipos de erros

- Uma forma de ver:
  - Erros de sintaxe
  - Erros de semântica
  - Erros de lógica
- Além disso:
  - Erros em tempo de compilação
  - Erros em tempo de execução

Erros de Sintaxe

Exemplos

- Erros associados ao fato de que a sintaxe da linguagem não está sendo respeitada
- Quase sempre são erros de compilação

```
#include <iostream>

int main() {
  st::cout << "oi";
  return 0;
}
```

Falta a letra ‘d’ (em ‘std’), além de fechar o parênteses.

Erros de Semântica

Exemplos

- Erros associados a um uso indevido de algumas declarações do programa
- Podem ou não gerar um erro/warning

```
int main() {
  int i;
  i++;
  return 0;
}
```

- Variável não inicializada
- Ocorre apenas um Warning

```
int main() {
  int a = “hello”;
  return 0;
}
```

- Atribuição incorreta de tipo

Erros de Lógica

- Não temos mais a ajuda do compilador
- Erros associados ao fato de que a especificação (comportamento desejado) não é respeitada
  - Exemplo: erros de cálculo

Erros de Lógica

Exemplo

```
#include <iostream>
#include <cmath>

int fatorial(int num) {
  int fat = 0;
  for (int i = 1; i <= num; i++)
    fat = fat * i;
  return fat;
}

double series(double x, int n) {
  double valor = 0.0;
  double xpow = 1;
  for (int k = 0; k <= n; k++) {
    valor += xpow / fatorial(k);
    xpow = xpow * x;
  }
  return valor;
}
```

Erros de Lógica

Não temos mais ajuda do compilador

```
#include <iostream>
#include <cmath>

int fatorial(int num) {
  int fat = 0;
  for (int i = 1; i <= num; i++)
    fat = fat * i;
  return fat;
}

double series(double x, int n) {
  double valor = 0.0;
  double xpow = 1;
  for (int k = 0; k <= n; k++) {
    valor += xpow / fatorial(k);
    xpow = xpow * x;
  }
  return valor;
}
```

Fatorial sempre zero

Depuração

Formalizando… Tipos de Erros - Momento da detecção

- Tempo de compilação
  - Erros de sintaxe e erros semânticos estáticos são indicados pelo próprio compilador
- Tempo de execução
  - Erros semânticos dinâmicos e erros lógicos, dificilmente podem ser detectados pelo compilador
  - É necessário depurar o código!

Depuração

Tipos de Erros - Momento da detecção

O programa vai compilar,

e o problema só será detectado durante a execução do mesmo.

```
int main() {
  int a, b, x;
  a = 10;
  b = 0;
  x = a / b;

  return 0;
}
```

Passos para Depurar o Código

- Reproduzir o problema
  - Determinar condições / estabilizar
- Achar o local do erro
  - Dados ⇨ Hipótese ⇨ Repetir
- Fazer a correção do erro
  - Considerar a real razão do problema
- Avaliar a solução e procurar erros similares
  - Testar novamente; possíveis efeitos colaterais

Dicas Gerais

- Entenda as mensagens de erro
- Pense antes de escrever
- Procure por problemas comuns
- Dividir para conquistar
- Mostre o valor de variáveis importantes
- Utilize um depurador
- Concentre-se em mudanças recentes

Como NÃO deve ser feito!

- Encontrar defeitos adivinhando (na sorte)
- Fazer alterações aleatórias até funcionar
- Não fazer um backup do original e não manter um histórico das alterações feitas
- Corrigir o erro com a solução mais óbvia sem entender a razão do problema
  - Se o erro sumiu, tudo ok! Problema resolvido! :(

GNU Project Debugger - GDB

- **GNU Debugger**
  - Depurador "padrão" de C/C++
- Permite acompanhar o que está acontecendo dentro do programa enquanto é executado
  - Executar programa
  - Adicionar breakpoints
  - Analisar o código uma linha de cada vez
  - Verificar valores de variáveis durante a execução

GDB

- Pode ser utilizado pela linha de comando
  - Utilizar a flag de compilação "-g" (debug)
- Ou com uma GUI
  - http://gdbgui.com
- Ou em uma IDE
  - CodeBlocks e Visual Studio
- Até online
  - https://www.onlinegdb.com/

Erros de Memória

- Memory Leaks
  - Código sem delete
- Acessos inválidos
  - Acesso para null
  - Acesso para memória desalocada
- Arquivos abertos
  - Arquivos sem close
- Para erros de memória podemos fazer uso de ferramentas...

Valgrind

- Ferramentas que ajudam a detectar erros associados ao gerenciamento de memória
  - Memory leaks, erros de alocação ou desalocação, ...
  - Comum em ambientes Unix
- Ferramentas auxiliares
  - DrMemory
    - Disponível para Windows/Linux/Mac
  - Valkyrie
  - Dicas de como instalar:
    - https://github.com/flaviovdf/programacao-2/tree/master/valgriddrmem