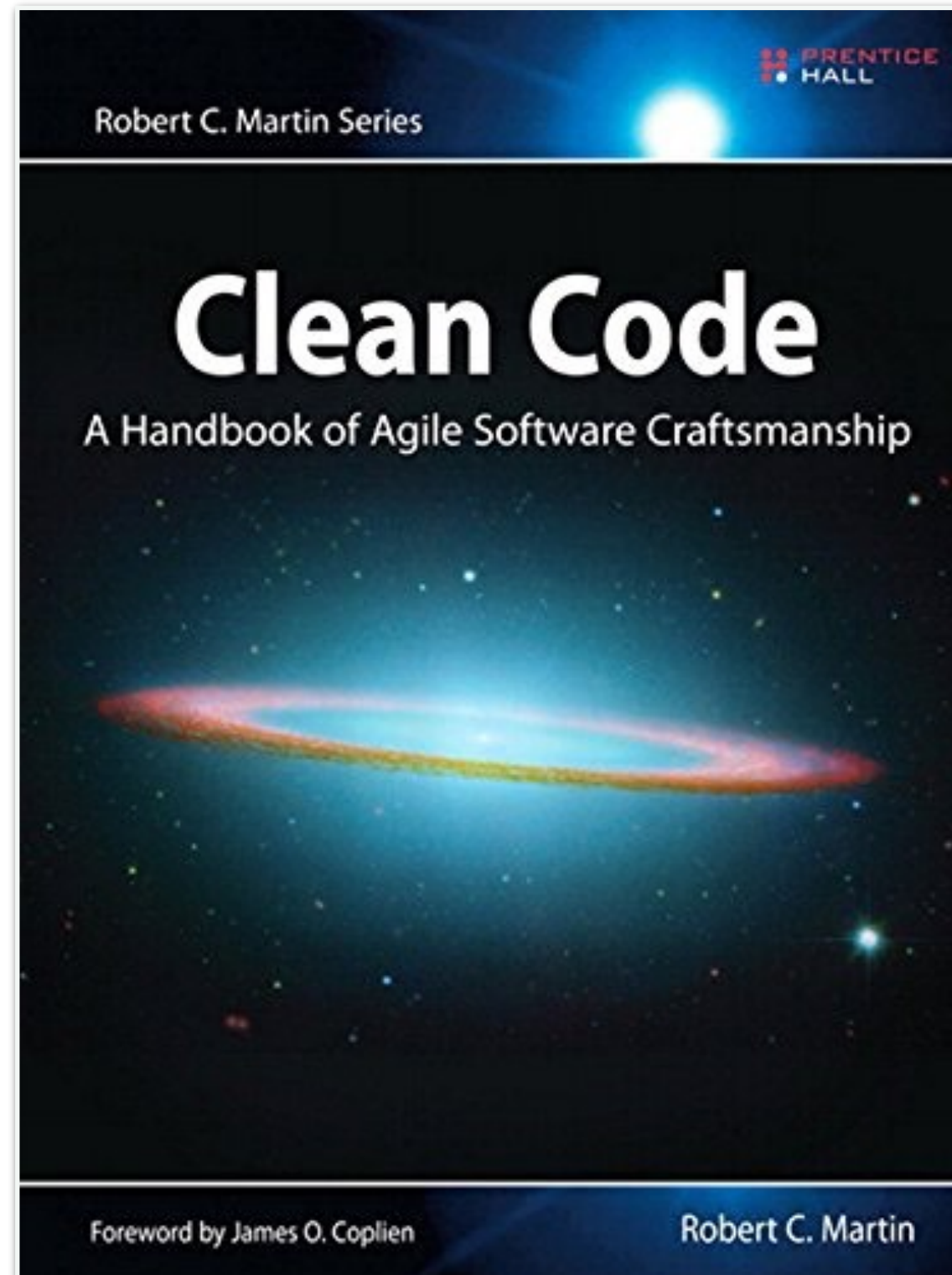


Engenharia de Software II

Código limpo - parte 1:
legibilidade, comentários e formatação

Prof. André Hora
DCC/UFMG
2019.1



“You are reading this book for two reasons. First, you are a programmer. Second, you want to be a better programmer. Good. We need better programmers.”

Código Limpo

- Boas práticas de programação
- Como escrever código com qualidade
- Diferenças entre código bom e ruim
- Como transformar código ruim em bom
- Exemplos práticos

O que é código limpo?

```
for (int j=0; j<34; j++) {  
    s += (t[j]*4)/5;  
}
```

vs.

```
int realDaysPerIdealDay = 4;  
const int WORK_DAYS_PER_WEEK = 5;  
int sum = 0;  
for (int j=0; j < NUMBER_OF_TASKS; j++) {  
    int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;  
    int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);  
    sum += realTaskWeeks;  
}
```

O que é código limpo?

*“Clean code is **simple** and **direct**. Clean code reads like well-written prose. Clean code never obscures the designer’s intent but rather is full of abstractions and straightforward lines of control.”*

Grady Booch, criador da UML



O que é código limpo?

*“I like my code to be **elegant** and **efficient**. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well.”*

Bjarne Stroustrup, criador do C++



O que é código limpo?

*“Clean code can be **read**, and **enhanced** by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API.”*

Dave Thomas, idealizador do Eclipse



Agenda

- Legibilidade (cap 2)
- Comentários em código (cap 4)
- Formatação (cap 5)
- Funções (cap 3)
- Código externo (cap 8)
- Testes de unidade (cap 9)
- Classes (cap 10)

Agenda

- **Legibilidade (cap 2)**
- **Comentários em código (cap 4)**
- **Formatação (cap 5)**
- Funções (cap 3)
- Código externo (cap 8)
- Testes de unidade (cap 9)
- Classes (cap 10)

Legibilidad

Legibilidade

- Nomes são tudo em um software
- Nomeamos: variáveis, métodos, argumentos, classes, pacotes, arquivos, diretórios...


Dicas de Legibilidade

- Usar nomes que revelam a intenção
- Fazer distinção entre nomes
- Usar nomes pronunciáveis
- Usar nomes “buscáveis”

Usar nomes que revelam a intenção

- `int d; //tempo em dias`

Usar nomes que revelam a intenção

- `int d; //tempo em dias` 
- `int tempoEmDias;`
- `int diasDesdeACriacao;`
- `int diasDesdeAUltimaModificacao;`
- `int idadeDoArquivoEmDias;`



Usar nomes que revelam a intenção

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```


Usar nomes que revelam a intenção

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```

```
public List<int[]> getFlaggedCells() {  
    List<int[]> flaggedCells = new ArrayList<int[]>();  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```

Usar nomes que revelam a intenção

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```

```
public List<int[]> getFlaggedCells() {  
    List<int[]> flaggedCells = new ArrayList<int[]>();  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```

```
public List<Cell> getFlaggedCells() {  
    List<Cell> flaggedCells = new ArrayList<Cell>();  
    for (Cell cell : gameBoard)  
        if (cell.isFlagged())  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```

Usar nomes que revelam a intenção

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```



```
public List<int[]> getFlaggedCells() {  
    List<int[]> flaggedCells = new ArrayList<int[]>();  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```



```
public List<Cell> getFlaggedCells() {  
    List<Cell> flaggedCells = new ArrayList<Cell>();  
    for (Cell cell : gameBoard)  
        if (cell.isFlagged())  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```



Fazer distinção entre nomes

```
public static void copyChars(char a1[], char a2[]) {  
    for (int i = 0; i < a1.length; i++) {  
        a2[i] = a1[i];  
    }  
}
```

Fazer distinção entre nomes

source e destination



```
public static void copyChars(char a1[], char a2[]) {  
    for (int i = 0; i < a1.length; i++) {  
        a2[i] = a1[i];  
    }  
}
```

Usar nomes pronunciáveis

```
class DtaRcrd102 {  
    private Date genymdhms;  
    private Date modymdhms;  
    private final String pszqint = "102";  
    /* ... */  
};
```

Usar nomes pronunciáveis

```
class DtaRcrd102 {  
    private Date genymdhms;  
    private Date modymdhms;  
    private final String pszqint = "102";  
    /* ... */  
};
```

```
class Customer {  
    private Date generationTimestamp;  
    private Date modificationTimestamp;  
    private final String recordId = "102";  
    /* ... */  
};
```

Usar nomes pronunciáveis

```
class DtaRcrd102 {  
    private Date genymdhms;  
    private Date modymdhms;  
    private final String pszqint = "102";  
    /* ... */  
};
```



```
class Customer {  
    private Date generationTimestamp;  
    private Date modificationTimestamp;  
    private final String recordId = "102";  
    /* ... */  
};
```




Usar nomes “buscáveis”

```
for (int j=0; j<34; j++) {  
    s += (t[j]*4)/5;  
}
```

Usar nomes “buscáveis”

```
for (int j=0; j<34; j++) {  
    s += (t[j]*4)/5;  
}
```

```
int realDaysPerIdealDay = 4;  
const int WORK_DAYS_PER_WEEK = 5;  
int sum = 0;  
for (int j=0; j < NUMBER_OF_TASKS; j++) {  
    int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;  
    int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);  
    sum += realTaskWeeks;  
}
```



Ambos os códigos produzem os mesmos resultados,
qual é o melhor?

Usar nomes “buscáveis”

```
for (int j=0; j<34; j++) {  
    s += (t[j]*4)/5;  
}
```



```
int realDaysPerIdealDay = 4;  
const int WORK_DAYS_PER_WEEK = 5;  
int sum = 0;  
for (int j=0; j < NUMBER_OF_TASKS; j++) {  
    int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;  
    int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);  
    sum += realTaskWeeks;  
}
```



Ambos os códigos produzem os mesmos resultados,
qual é o melhor?

Comentários em código

Comentários

- Não existe nada mais útil do que um bom comentário
- Não existe nada pior do que um comentário ruim, redundante, desnecessário...
- O propósito dos comentários é compensar a falha de nos expressarmos em código

Qual das opções abaixo é a melhor?

```
// Check to see if the employee is eligible for full benefits  
if ((employee.flags & HOURLY_FLAG) &&  
    (employee.age > 65))
```

```
if (employee.isEligibleForFullBenefits())
```

Qual das opções abaixo é a melhor?

```
// Check to see if the employee is eligible for full benefits  
if ((employee.flags & HOURLY_FLAG) &&  
    (employee.age > 65))
```

```
if (employee.isEligibleForFullBenefits())
```



Muitas vezes, o código pode ser auto explicativo

Comentários relevantes

```
// Returns an instance of the Responder being tested.  
protected abstract Responder responderInstance();
```

```
// format matched kk:mm:ss EEE, MMM dd, yyyy  
Pattern timeMatcher = Pattern.compile(  
    "\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

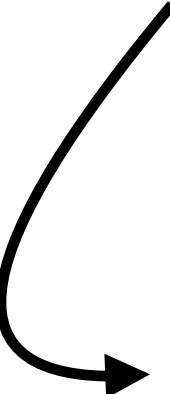

Comentários relevantes

```
// Returns an instance of the Responder being tested.  
protected abstract Responder responderInstance();
```



Melhor: mudar o nome do método para `responderBeingTested`

```
// format matched kk:mm:ss EEE, MMM dd, yyyy  
Pattern timeMatcher = Pattern.compile(  
    "\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```



Melhor: código poderia ser movido para uma classe que trata datas e tempo. Nesse caso, nem precisaria de comentário!

Comentários redundantes

```
/** The name. */  
private String name;  
  
/** The version. */  
private String version;  
  
/** The licenceName. */  
private String licenceName;  
  
/** The version. */  
private String info;
```

```
/** The day of the month. */  
private int dayOfMonth;
```

```
/**  
 * Default constructor.  
 */  
protected AnnualDateRule() {  
}
```

```
/**  
 * Returns the day of the month.  
 *  
 * @return the day of the month.  
 */  
public int getDayOfMonth() {  
    return dayOfMonth;  
}
```

Comentários redundantes

```
public abstract class ContainerBase
    implements Container, Lifecycle, Pipeline,
    MBeanRegistration, Serializable {

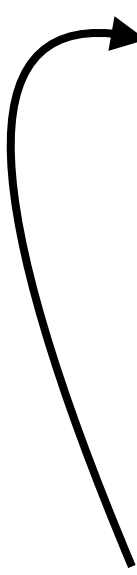
    /**
     * The processor delay for this component.
     */
    protected int backgroundProcessorDelay = -1;

    /**
     * The lifecycle event support for this component.
     */
    protected LifecycleSupport lifecycle =
        new LifecycleSupport(this);

    /**
     * The container event listeners for this Container.
     */
    protected ArrayList listeners = new ArrayList();

    /**
     * The Loader implementation with which this Container is
     * associated.
     */
    protected Loader loader = null;
```

Comentários desatualizados



```
MockRequest request;
private final String HTTP_DATE_REGEXP =
    "[SMTWF][a-z]{2}\\,\\s[0-9]{2}\\s[JFMASOND][a-z]{2}\\s"+
    "[0-9]{4}\\s[0-9]{2}\\:[0-9]{2}\\:[0-9]{2}\\sGMT";
private Response response;
private FitNesseContext context;
private FileResponder responder;
private Locale saveLocale;
// Example: "Tue, 02 Apr 2003 22:18:49 GMT"
```

Código Comentado

```
InputStreamResponse response = new InputStreamResponse();  
response.setBody(formatter.getResultStream(), formatter.getByteCount());  
// InputStream resultsStream = formatter.getResultStream();  
// StreamReader reader = new StreamReader(resultsStream);  
// response.setContent(reader.read(formatter.getByteCount()));
```

Código Comentado

```
while (!stack.isEmpty()) {  
    State current = stack.pop();  
  
    // int theDepth = current.getDepth();  
    // if(theDepth > maxDepthSearched)  
    //     maxDepthSearched = theDepth;  
  
    if (current.isGoal(maze)){  
        // TODO update the maze if a solution found  
        //while(current != null){  
        while(current.getParent() != null){  
            if(!current.isGoal(maze))  
                maze.setOneSquare(current.getSquare(), '.');  
            current = current.getParent();  
        }  
        return true;  
    }  
}
```

*Commented-out code:
Código “removido”
através de comentário*

Exercício

- Escreva 4 desvantagens de código comentado?

Exercício

- Desvantagens de código comentado?

1. Código comentado é uma distração e pode confundir o programador
2. Código comentado sempre levanta mais questões do que respostas
3. Programadores esquecerão rapidamente o quão relevante é o código comentado
4. Código comentado é uma forma errada de controle de versão, uma vez que ferramentas como Git e SVN são indiscutivelmente a solução apropriada
5. O simples fato do programador buscar entender a razão por trás do código comentado pode tomar muito tempo
6. Código comentado torna-se defasado ao longo do tempo uma vez que o sistema continua a evoluir enquanto o trecho de código comentado permanece estático

Formatação

Formatação

- Formatação é **importante**
- Formatação de código trata de **comunicação**
- Comunicação é fundamental para desenvolvedores profissionais

Exercício

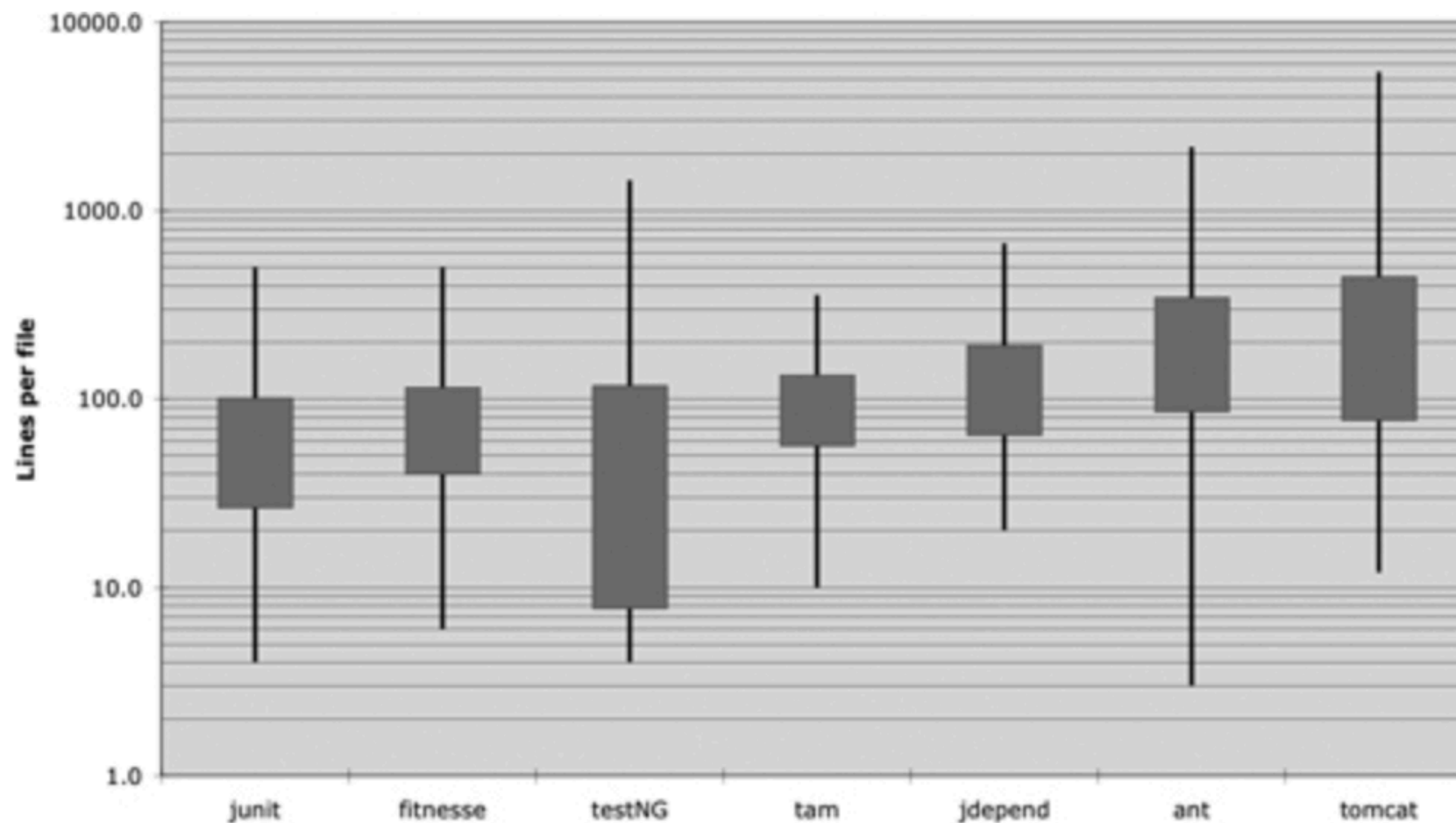
- O que podemos fazer para melhorar a formatação de um código?

Dicas de Formatação

- Formatação vertical
- Separação de conceitos
- Funções dependentes
- Afinidade conceitual
- Formatação horizontal
- Indentação

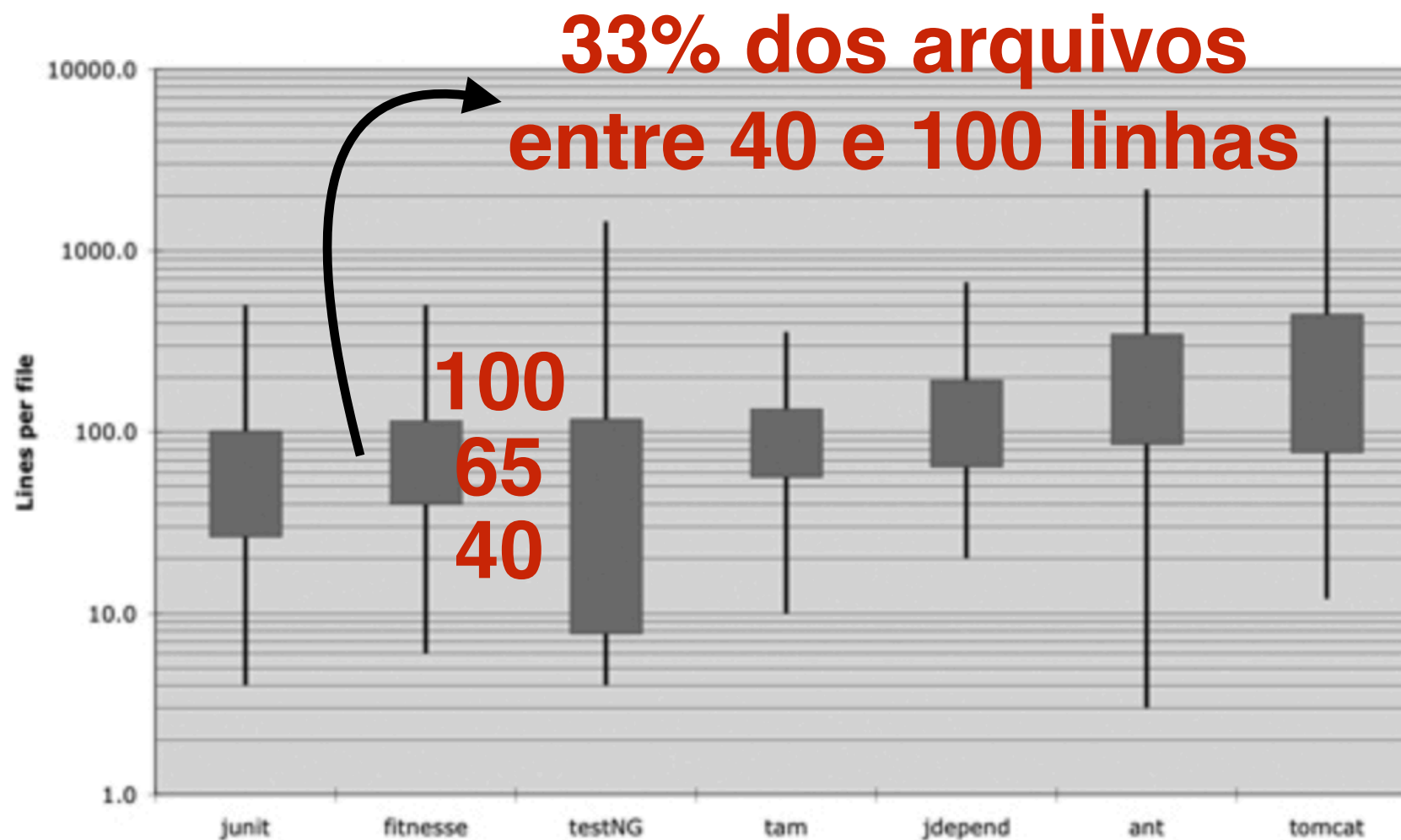
Formatação vertical

- O quão grande deve ser um arquivo?



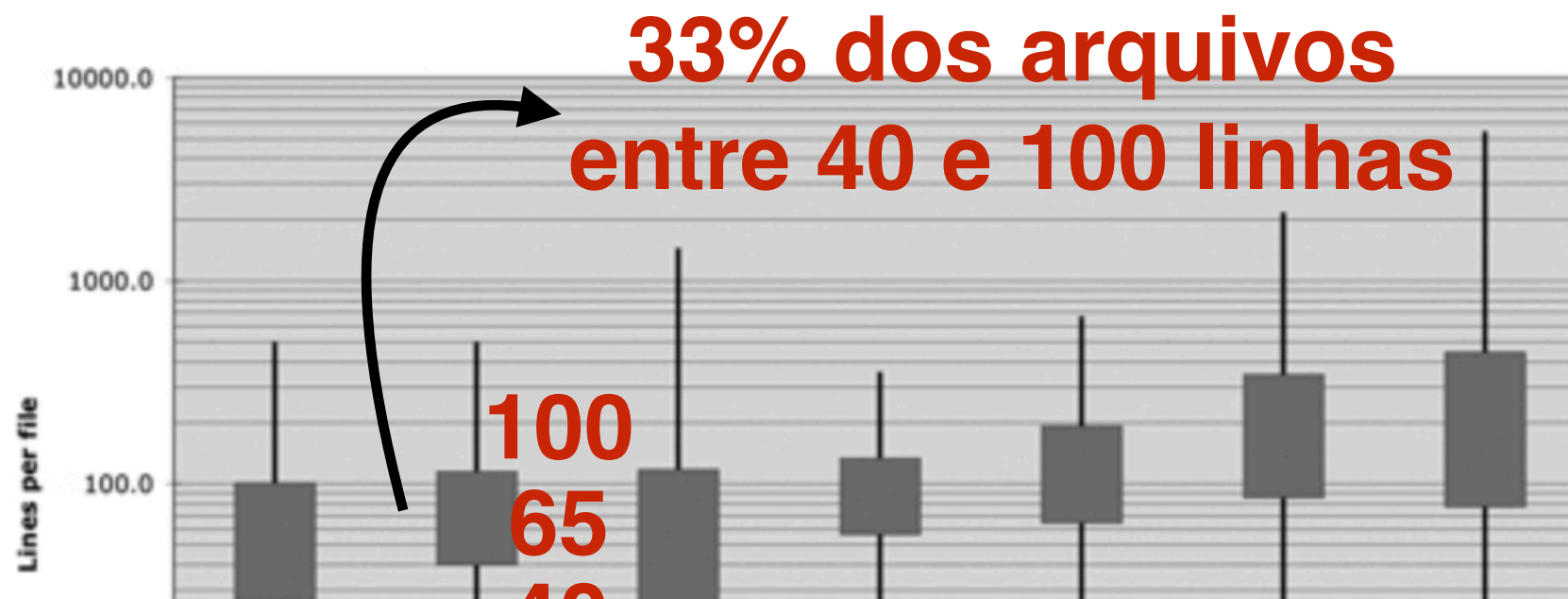
Formatação vertical

- O quão grande deve ser um arquivo?



Formatação vertical

- O quão grande deve ser um arquivo?



É possível construir sistemas reais com arquivos não muito longos

Formatação vertical

- Comparação: código x jornal
- Artigos são lidos verticalmente
- No topo: manchetes
- Primeiro parágrafo: sinopse
- Abaixo: detalhes



Formatação vertical

- Idealmente, um arquivo de código deve ser igual a um artigo de jornal
- Nome: simples e explicativo
- Topo: funções de alto nível
- Mais abaixo: funções com mais detalhes

Separação de conceitos

- Cada grupo de linhas deve representar um conceito
- Regra extremamente simples

```
package fitnessse.wikitext.widgets;
import java.util.regex.*;

public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?'''";
    private static final Pattern pattern = Pattern.compile("'''(.+?)'''"
        Pattern.MULTILINE + Pattern.DOTALL);
    public BoldWidget(ParentWidget parent, String text) throws Exception {
        super(parent);
        Matcher match = pattern.matcher(text);
        match.find();
        addChildWidgets(match.group(1));
    }
    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```

```
package fitnessse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?'''";
    private static final Pattern pattern = Pattern.compile("'''(.+?)'"
        Pattern.MULTILINE + Pattern.DOTALL
    );

    public BoldWidget(ParentWidget parent, String text) throws Except.
        super(parent);
        Matcher match = pattern.matcher(text);
        match.find();
        addChildWidgets(match.group(1));
    }

    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```

Separação de conceitos

- Cada grupo de linhas deve representar um conceito
- Regra extremamente simples

```
package fitnessse.wikitext.widgets;
import java.util.regex.*;
public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?'";
    private static final Pattern pattern = Pattern.compile("'''.+?'",
        Pattern.MULTILINE + Pattern.DOTALL);
    public BoldWidget(ParentWidget parent, String text) throws Exception {
        super(parent);
        Matcher match = pattern.matcher(text);
        match.find();
        addChildWidgets(match.group(1));
    }
    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```



```
package fitnessse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
    public static final String REGEXP = "'''.+?'";
    private static final Pattern pattern = Pattern.compile("'''.+?'",
        Pattern.MULTILINE + Pattern.DOTALL
    );

    public BoldWidget(ParentWidget parent, String text) throws Except.
        super(parent);
        Matcher match = pattern.matcher(text);
        match.find();
        addChildWidgets(match.group(1));
    }

    public String render() throws Exception {
        StringBuffer html = new StringBuffer("<b>");
        html.append(childHtml()).append("</b>");
        return html.toString();
    }
}
```



Funções dependentes

- Se uma função chama outra, elas devem estar próximas (chamador acima da chamada)
- Fornece ao código um fluxo natural

```

public class WikiPageResponder implements SecureResponder {
    protected WikiPage page;
    protected PageData pageData;
    protected String pageTitle;
    protected Request request;
    protected PageCrawler crawler;

    public Response makeResponse(FitNesseContext context, Request request)
        throws Exception {
        String pageName = getPageNameOrDefault(request, "FrontPage");
        loadPage(pageName, context);
        if (page == null)
            return notFoundResponse(context, request);
        else
            return makePageResponse(context);
    }

    private String getPageNameOrDefault(Request request, String defaultPageName)
    {
        String pageName = request.getResource();
        if (StringUtil.isBlank(pageName))
            pageName = defaultPageName;

        return pageName;
    }

    protected void loadPage(String resource, FitNesseContext context)
        throws Exception {
        WikiPagePath path = PathParser.parse(resource);
        crawler = context.root.getPageCrawler();
        crawler.setDeadEndStrategy(new VirtualEnabledPageCrawler());
        page = crawler.getPage(context.root, path);
        if (page != null)
            pageData = page.getData();
    }

    private Response notFoundResponse(FitNesseContext context, Request request)
        throws Exception {
        return new NotFoundResponder().makeResponse(context, request);
    }
}

```

```

public class WikiPageResponder implements SecureResponder {
    protected WikiPage page;
    protected PageData pageData;
    protected String pageTitle;
    protected Request request;
    protected PageCrawler crawler;

    public Response makeResponse(FitNesseContext context, Request request)
        throws Exception {
        1 String pageName = getPageNameOrDefault(request, "FrontPage");
        2 loadPage(pageName, context);
        if (page == null)
            3 return notFoundResponse(context, request);
        else
            return makePageResponse(context);
    }

    private String getPageNameOrDefault(Request request, String defaultPageName)
    {
        String pageName = request.getResource();
        1 if (StringUtil.isBlank(pageName))
            pageName = defaultPageName;

        return pageName;
    }

    protected void loadPage(String resource, FitNesseContext context)
        throws Exception {
        2 WikiPagePath path = PathParser.parse(resource);
        crawler = context.root.getPageCrawler();
        crawler.setDeadEndStrategy(new VirtualEnabledPageCrawler());
        page = crawler.getPage(context.root, path);
        if (page != null)
            pageData = page.getData();
    }

    3 private Response notFoundResponse(FitNesseContext context, Request request)
        throws Exception {
        return new NotFoundResponder().makeResponse(context, request);
    }
}

```

Afinidade conceitual

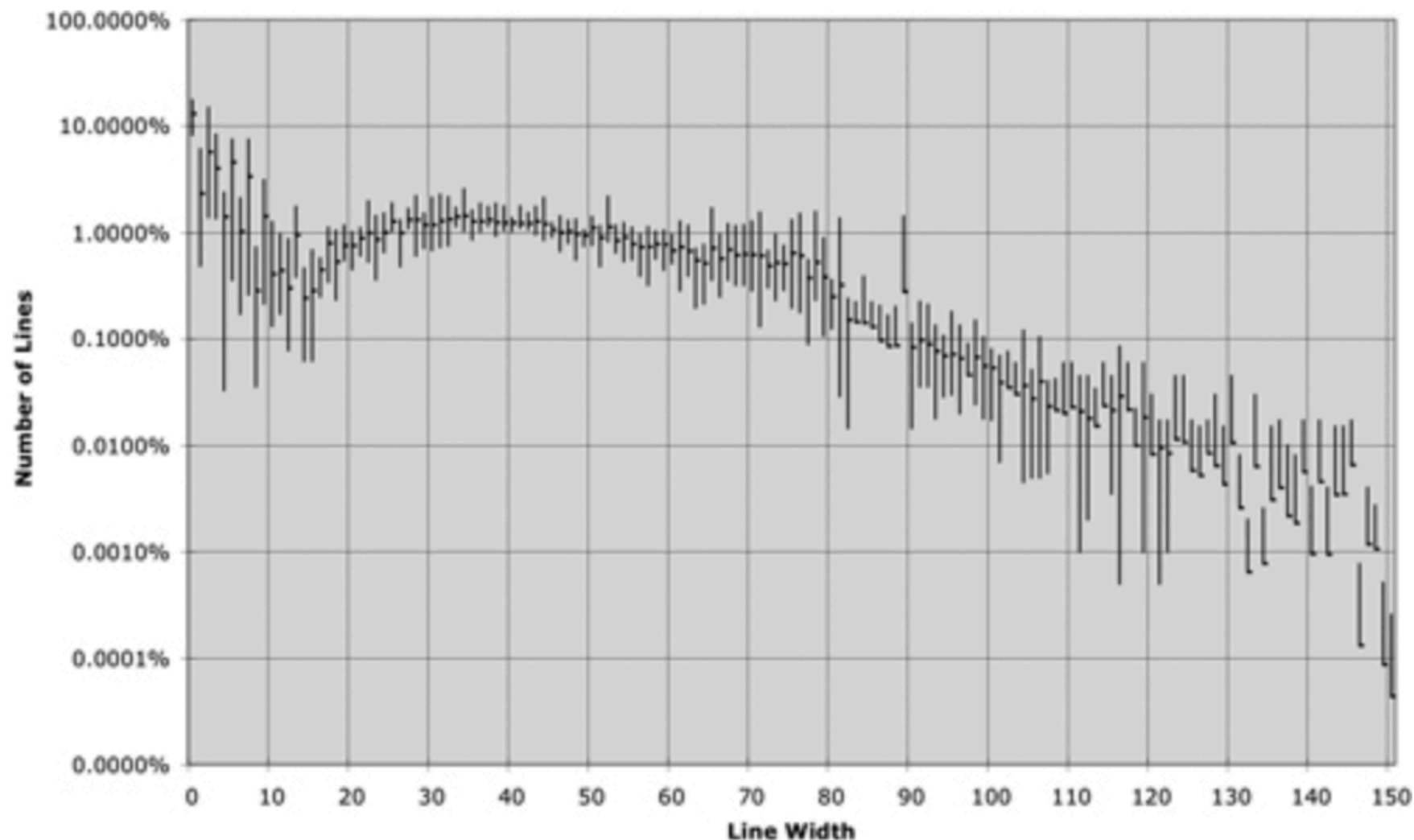
- Dependência direta ou operações similares

```
public class Assert {  
    static public void assertTrue(String message, boolean condition) {  
        if (!condition)  
            fail(message);  
    }  
  
    static public void assertTrue(boolean condition) {  
        assertTrue(null, condition);  
    }  
  
    static public void assertFalse(String message, boolean condition) {  
        assertTrue(message, !condition);  
    }  
  
    static public void assertFalse(boolean condition) {  
        assertFalse(null, condition);  
    }  
    ...  
}
```

**Variações
da mesma
tarefa básica**

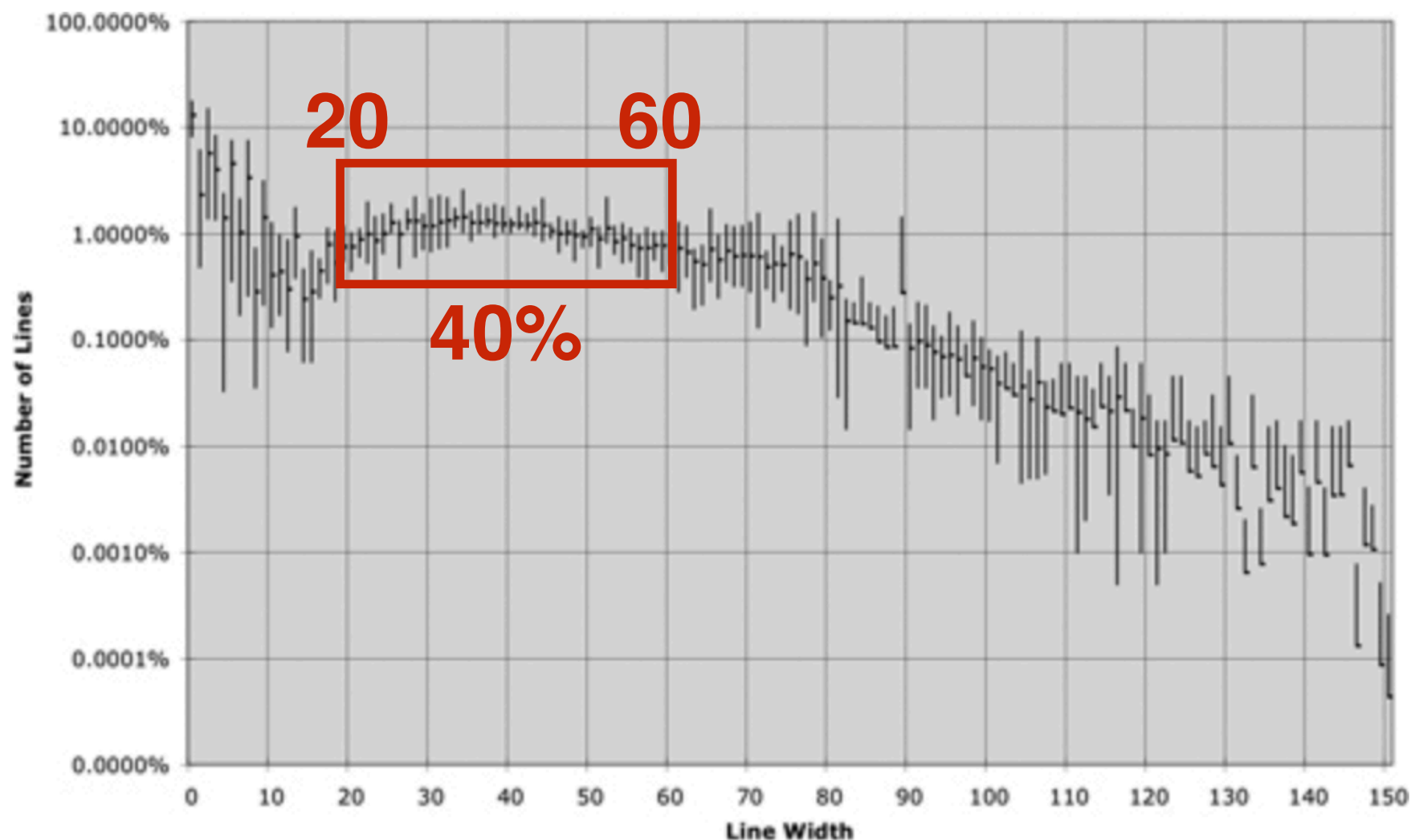
Formatação horizontal

- O quão comprida uma linha deve ser?



Formatação horizontal

- O quão comprida uma linha deve ser?



Formatação horizontal

- O quão comprida uma linha deve ser?



Indentação

- Código fonte é hierárquico
 - Classes, métodos, blocos de código, etc
 - Cada nível é um escopo, onde variáveis podem ser declaradas e utilizadas
- Indentação: torna os níveis visíveis
 - Permite visualizar facilmente escopos de ifs, whiles, fors
- Sem indentação: códigos tornam-se ilegíveis para humanos

```
public class FitNesseServer implements SocketServer { private FitNesseCont
context; public FitNesseServer(FitNesseContext context) { this.context =
context; } public void serve(Socket s) { serve(s, 10000); } public void
serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = ne
FitNesseExpediter(s, context);
sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); }
catch(Exception e) { e.printStackTrace(); } } }
```

```

public class FitNesseServer implements SocketServer { private FitNesseCont
context; public FitNesseServer(FitNesseContext context) { this.context =
context; } public void serve(Socket s) { serve(s, 10000); } public void
serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = ne
FitNesseExpediter(s, context);
sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); }
catch(Exception e) { e.printStackTrace(); } } }

```

```

public class FitNesseServer implements SocketServer {
    private FitNesseContext context;

    public FitNesseServer(FitNesseContext context) {
        this.context = context;
    }

    public void serve(Socket s) {
        serve(s, 10000);
    }

    public void serve(Socket s, long requestTimeout) {
        try {
            FitNesseExpediter sender = new FitNesseExpediter(s, context);
            sender.setRequestParsingTimeLimit(requestTimeout);
            sender.start();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
public class FitNesseServer implements SocketServer { private FitNesseCont
context; public FitNesseServer(FitNesseContext context) { this.context =
context; } public void serve(Socket s) { serve(s, 10000); } public void
serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = ne
FitNesseExpediter(s, context);
sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); }
catch(Exception e) { e.printStackTrace(); } } }
```



```
public class FitNesseServer implements SocketServer {
    private FitNesseContext context;

    public FitNesseServer(FitNesseContext context) {
        this.context = context;
    }

    public void serve(Socket s) {
        serve(s, 10000);
    }

    public void serve(Socket s, long requestTimeout) {
        try {
            FitNesseExpediter sender = new FitNesseExpediter(s, context);
            sender.setRequestParsingTimeLimit(requestTimeout);
            sender.start();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

