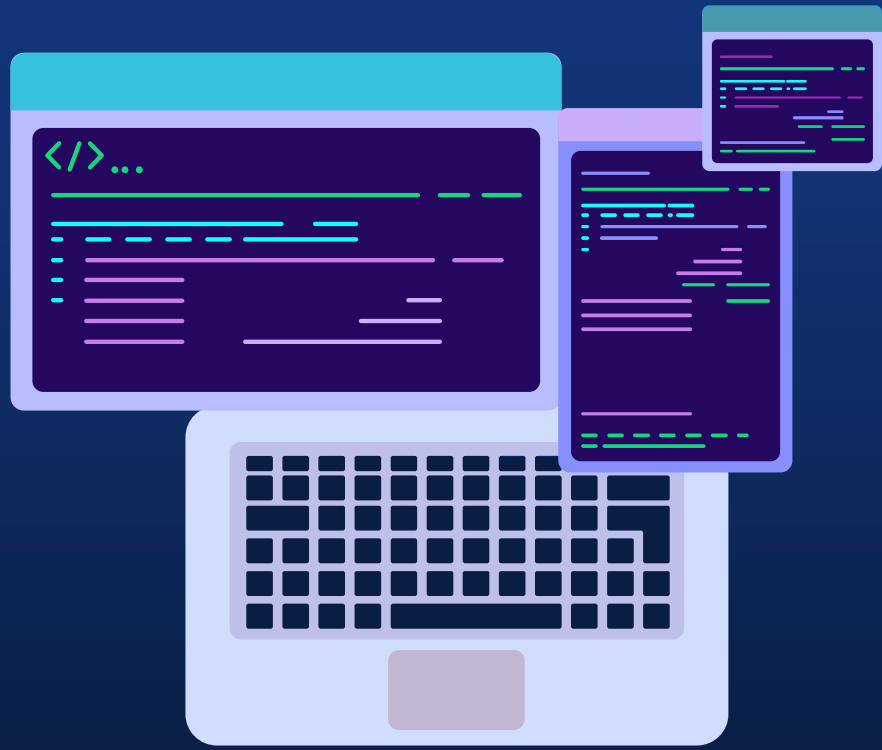
Software Design and Important concepts



Mentor: Einar Rocha

CONTENT

01
OOP Pillars

Inheritance, Polymorphism
Encapsulation, Abstraction

02
Clean Code

Meaningful Names,
Functions, Unit test
Code Smells...

03
SOLID

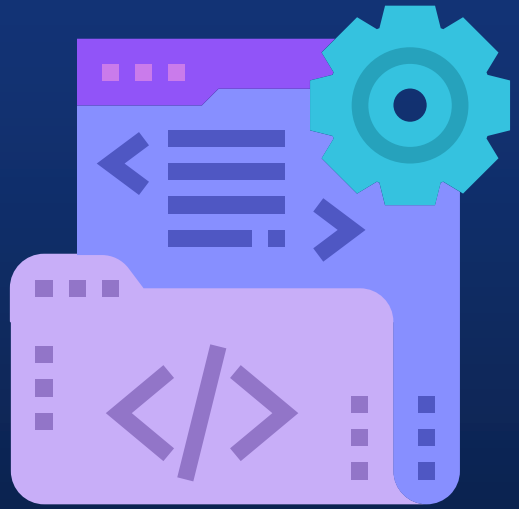
Single Responsiblity
Open closed
Liskov Substitution
Interface Segregation
Dependency Inversion

04
Design patterns

Singleton, Factory Method
Strategy, Observer
Builder...

02

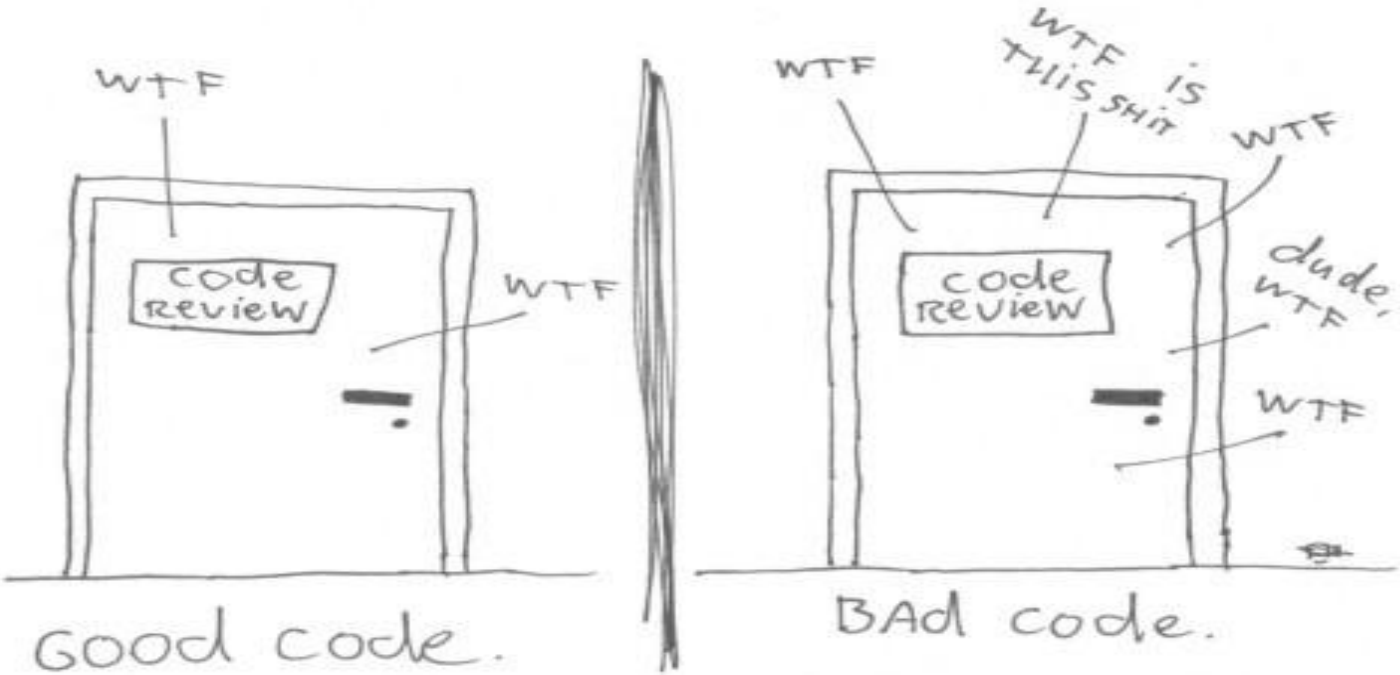
Clean Code



The Goals of Software Design

- To allow us to write software that is as helpful as possible.
- To allow our software to continue to be as helpful as possible.
- To design systems that can be created and maintained as easily as possible by their programmers

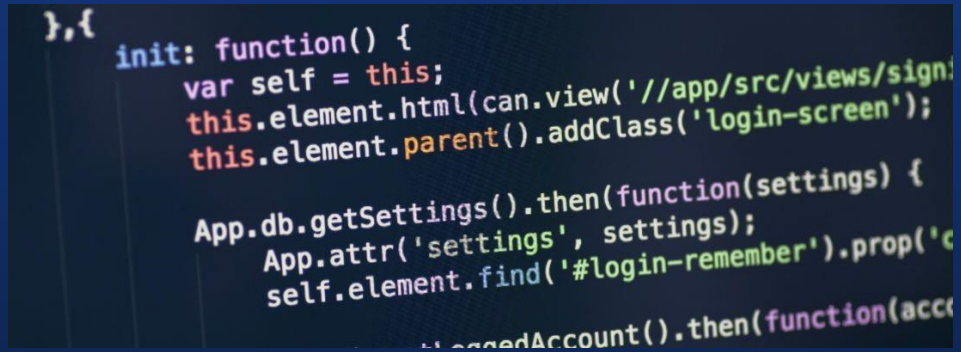
The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



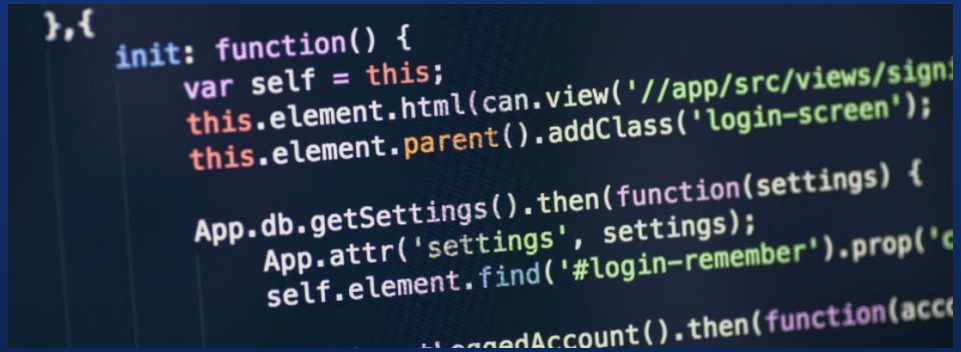
Agenda

- **Formatting** — Vertical, Horizontal...
- **Error Handling** — Use Exceptions, define, avoid null...

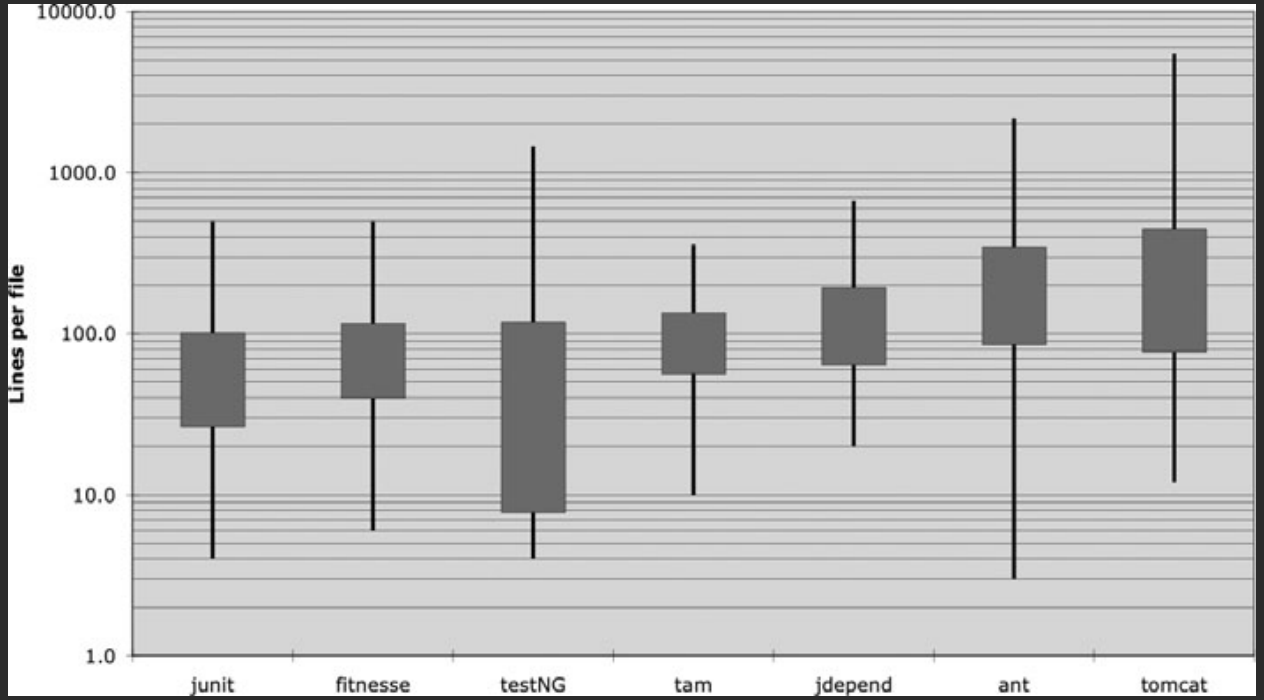
Formatting


},{
init: function() {
var self = this;
this.element.html(can.view('//app/src/views/signi
this.element.parent().addClass('login-screen');
App.db.getSettings().then(function(settings) {
App.attr('settings', settings);
self.element.find('#login-remember').prop('c
loggedAccount().then(function(acc

Purpose?


},{
init: function() {
var self = this;
this.element.html(can.view('//app/src/views/signi
this.element.parent().addClass('login-screen');
App.db.getSettings().then(function(settings) {
App.attr('settings', settings);
self.element.find('#login-remember').prop('c
gedAccount().then(function(acc

Vertical Formatting



Vertical Openness Between Concepts

```
package fitnesse.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();
  }
}
```

Vertical Openness Between Concepts

```
package fitnesse.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();
  }
}
```

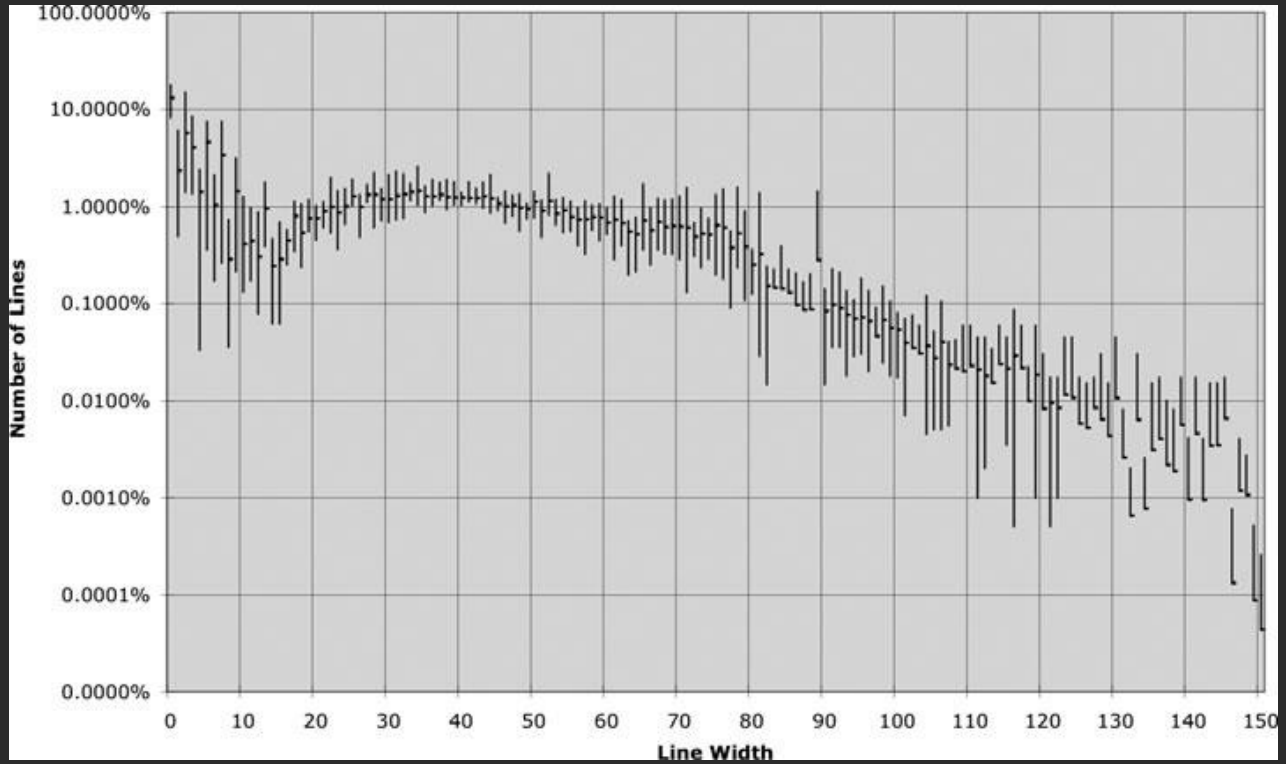
Vertical Density

```
public class ReporterConfig {
    /**
     * The class name of the reporter listener
     */
    private String className;

    /**
     * The properties of the reporter listener
     */
    private List<Property> properties = new ArrayList<Property>();

    public void addProperty(Property property) {
        properties.add(property);
    }
```

Vertical Density

```
public class ReporterConfig {
    private String m_className;
    private List<Property> m_properties = new ArrayList<Property>();

    public void addProperty(Property property) {
        m_properties.add(property);
    }
```

Horizontal Formatting



Indentation

```
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(); } } }
```

Indentation

```
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();
    }
  }
}
```

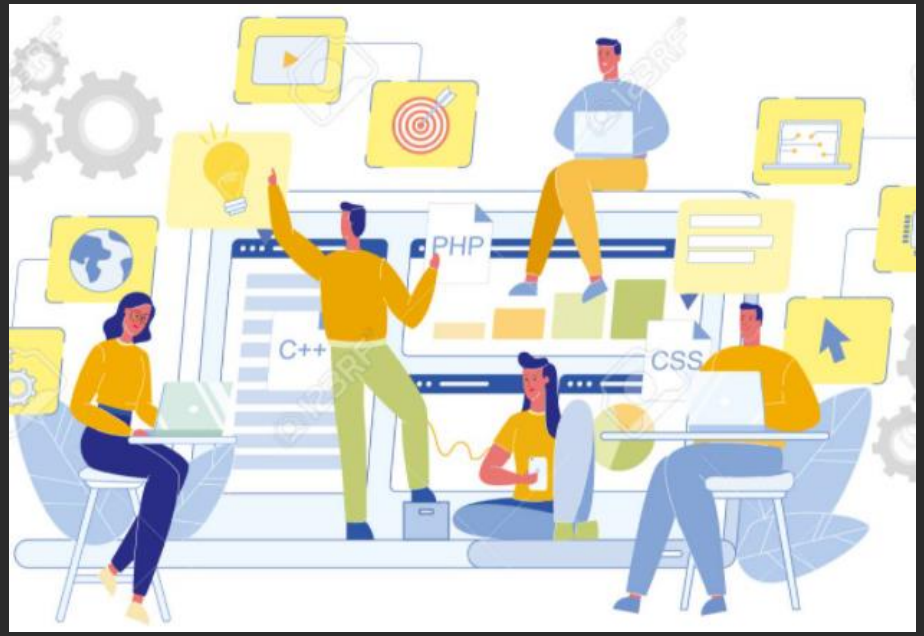
Breaking Indentation

```
public class CommentWidget extends TextWidget
{
    public static final String REGEXP = "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";

    public CommentWidget(ParentWidget parent, String text){super(parent, text);}
    public String render() throws Exception {return ""; }
}
```

Breaking Indentation

```
public class CommentWidget extends TextWidget {
   public static final String REGEXP = "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";

   public CommentWidget(ParentWidget parent, String text) {
      super(parent, text);
   }

   public String render() throws Exception {
      return "";
   }
}
```

Team Rules



Error Handling

Use Exceptions Rather Than Return Codes

```
public class DeviceController {
   public void sendShutDown() {
      DeviceHandle handle = getHandle(DEV1);
      // Check the state of the device
      if (handle != DeviceHandle.INVALID) {
         // Save the device status to the record field
         retrieveDeviceRecord(handle);
         // If not suspended, shut down
         if (record.getStatus() != DEVICE_SUSPENDED) {
            pauseDevice(handle);
            clearDeviceWorkQueue(handle);
            closeDevice(handle);
         } else {
            logger.log("Device suspended. Unable to shut down");
         }
      } else {
         logger.log("Invalid handle for: " + DEV1.toString());
      }
   }
...
}
```

Use Exceptions Rather Than Return Codes

```
public class DeviceController {

   public void sendShutDown() {
      try {
         tryToShutDown();
      } catch (DeviceShutDownError e) {
         logger.log(e);
      }
   }

   private void tryToShutDown() throws DeviceShutDownError {
      DeviceHandle handle = getHandle(DEV1);
      DeviceRecord record = retrieveDeviceRecord(handle);
      pauseDevice(handle);
      clearDeviceWorkQueue(handle);
      closeDevice(handle);
   }
}
```

Define Exception Classes in Terms of a Caller's Needs

```
ACMEPort port = new ACMEPort(12);

try {
   port.open();
   } catch (DeviceResponseException e) {
   reportPortError(e);
   logger.log("Device response exception", e);
   } catch (ATM1212UnlockedException e) {
   reportPortError(e);
   logger.log("Unlock exception", e);
   } catch (GMXError e) {
   reportPortError(e);
   logger.log("Device response exception");
   } finally {
   …
   }
```

Define Exception Classes in Terms of a Caller's Needs

```
LocalPort port = new LocalPort(12);
try {
   port.open();
} catch (PortDeviceFailure e) {
   reportError(e);
   logger.log(e.getMessage(), e);
} finally {
…
}
```

Define Exception Classes in Terms of a Caller's Needs

```
public class LocalPort {
   private ACMEPort innerPort;
   public LocalPort(int portNumber) {
      innerPort = new ACMEPort(portNumber);
   }
   public void open() {
      try {
         innerPort.open();
      } catch (DeviceResponseException e) {
         throw new PortDeviceFailure(e);
      } catch (ATM1212UnlockedException e) {
         throw new PortDeviceFailure(e);
      } catch (GMXError e) {
         throw new PortDeviceFailure(e);
      }
   }
```

Don’t Return Null

```
public void registerItem(Item item) {
    if (item != null) {
        ItemRegistry registry = peristentStore.getItemRegistry();
        if (registry != null) {
            Item existing = registry.getItem(item.getID());
            if (existing.getBillingPeriod().hasRetailOwner()) {
                existing.register(item);
            }
        }
    }
}
```

Don’t Return Null

```
List<Employee> employees = getEmployees();
if (employees != null) {
    for(Employee e : employees) {
        totalPay += e.getPay();
    }
}
```

Don’t Return Null

```
List<Employee> employees = getEmployees();
for(Employee e:employees){
   totalPay+=e.getPay();
}

public List<Employee> getEmployees(){
   if(..there are no employees..)
      return Collections.emptyList();
}
```

Don't Pass Null

```
public class MetricsCalculator {
   public double xProjection(Point p1, Point p2) {
      return (p2.x –p1.x) *1.5;
   }
}


public double xProjection(Point p1, Point p2) {
   if (p1 == null || p2 == null) {
      throw InvalidArgumentException(
            "Invalid argument for MetricsCalculator.xProjection");
   }
   return (p2.x –p1.x) *1.5;
}
```