• Software for process automation in the printing industry

• Since 1999

• Integrating data from various sources to drive automated press plate production

Björn Latte

@lattebl

www.wobe-systems.com

@wobeSystems
The age of the Monolith
Challenges of system integration and Industry 4.0

• Large number of systems and data formats

• Huge amount of sensors and actors

• Lack of system integration

• Lack of automation in data processing
“Loose systems last longer and work better“

John Gall – Systems Theorist
Titan Flow UI

http://www.jpaulmorrison.com/fbp/
Reasons for Clean Code

- Code is read more often than it is written
- Hard to understand code is hard to maintain
- Complicated code is hard to test
- Inconsistency will lead to bloat
Reasons for Clean Code - Technical Debt

![Diagram showing the relationship between Cost of Change (CoC) and Responsiveness to change over time. The diagram illustrates how Technical Debt increases as the Cost of Change increases, leading to a decrease in Responsiveness. Optimal CoC is indicated by a horizontal line at the top, while the optimal Responsiveness is shown at the right. The technical debt is represented by the area between the actual CoC and the optimal CoC over time.](https://via.placeholder.com/150)

(C) Urs Enzler www.bbv.ch

(CC BY 3.0 DE) Clean Code Cheat Sheet

(Urs Enzler www.bbv.ch)
Using Clean Code

ONE DOES NOT SIMPLY WRITE CLEAN CODE
What is Clean Code?

- Consistent use of coding style
- Use of meaningful naming
- Short functions (less than 100 lines)
- Small classes
- KISS
- DRY
- SOLID
The Human Factor

Writing clean code requires

Commitment

Openness

Trust
Code Reviews

**HOW TO MAKE A GOOD CODE REVIEW**

- Address code smells that are not covered by metrics or static analysis
- Establish common understanding
- Foster knowledge transfer
- Require constructive criticism
Test Driven Development (TDD)

- Write a minimal test first
- Write just as much code as to make it pass
- Write more test code
- Write more code to make it pass again
Benefits of Test Driven Development

• Instant refactoring while writing the code

• Tests allow later extension of functionality without breaking existing functionality

• Helps to keep focus and make classes and functions short (Single Responsibility)

• Reduces the: “Good enough” syndrome
"For a static analysis project to succeed developers must feel they benefit from and enjoy using it":

C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, C. Jaspan
Lessons from Building Static Analysis Tools at Google, Communications of the ACM, Vol. 51 No. 4, April 2018

Matt: “so now I wanna know why raising a string exception is bad. Like what should I be doing instead? Since it thinks it’s a problem. And so none of these really help me”

Jason: “... like I mentioned with ... it gives you so many warnings and sifting through them is so, arduous that whenever I just look at it I’m like ehhh forget this.”

B. Johnson, Y. Song, E. Murphy-Hill, R. Bowdidge,
Complexity before refactoring

21 Functions
Complexity after refactoring

8 Functions
Comparison of complexity distribution

Lines of complex code in ujotypes-c before refactoring

Lines of complex code in ujotypes-c after refactoring

Complexity Score

LoC

LoC
Detail comparison of a single function
Continuous Integration / Continuous Delivery

YOU SHALL NOT PASS

21

EMLS'19 Clean Code
Pipelines as code

```
stages:
  - check
  - build
  - test

splint:
  stage: check
tags:
  - docker
image: industrialdevops/ujoetypes_check_linux64:20180816
allow_failure: true
script:
  # stage artifacts from upstream build...
  - curl -O artifacts.zip --header "PRIVATE-TOKEN:$ARTIFACT_TOKEN" ?
  - unzip artifacts.zip
  # perform the check...
  - splint +show-summary

UJOTYP-33: add Variant functions for "Boolean" types

- 4 jobs from master in 46 seconds (queued for 2 seconds)

Pipeline | Jobs | Failed Jobs
---------|------|-------------
Check    |      |             
        | ✔    | ✔ splint    
Build    |      |             
        | ✔    | build       
Test     |      |             
        | ✔    | test        
```
Example Pipelines

**Build**
- build

**Test**
- test

**Check**
- flake8
- pylint

**Check**
- complexity_lib
- flawfinder
- splint_lib
- splint_test

**Build**
- build

**Test**
- test
Lack of visibility and helpfulness

FINAL RESULTS:

./test/common_test_tools.c:26: [4] (format) vfprintf:
   If format strings can be influenced by an attacker, they can be exploited (CWE-134). Use a constant for the format specification.
.
./src/ujoytypes.c:327: [2] (buffer) memcp:
   Does not check for buffer overflows when copying to destination (CWE-120). Make sure destination can always hold the source data.
.
./src/ujoytypes_float.c:69: [2] (buffer) char:
   Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.

Non-comment line ct:   >12
Average line score:    8
25%-ile score:         5 (75% in higher score procs)
50%-ile score:         7 (half in higher score procs)
75%-ile score:         13 (25% in higher score procs)
Highest score:         13 (ujot_create_ujoy_variant_from_ujo_element() in src/ujoytypes.c)
Job succeeded
Feedback to the Developer

• As early as possible:
  • Inside the Development Environment
  • Tools that run in the pipeline should also be available locally

• As visible as possible

• The Pipelines are the final Quality Gate
Take Aways

• Clean code is maintainable code
• Address the human factor of *getting clean*
• Static Code Analysis is an enabler for cleaner code
• Use automation for developer feedback and as a final quality gate
Topics for the Workshop

• Can Software be planned to be Long-Living?
• The “human factor“ in producing maintainable code.
• What is the best time to enforce Clean Code in a project?
• What needs to be done to enable / improve static code analysis tools to make their use helpful (or even fun)?
πάντα ῥεῖ [pánta reí] everything flows
Heraclitus 535 - 475 BC