Software Engineering Best Practices

TNG TECHNOLOGY CONSULTING

Maximilian Imgrund, Alexander Bartl

December 4, 2018

Who are we?

- Theoretical astrophysics, 2016
- USM - LMU München
- Business logic for a major telco
- PHP, Java
- maximilian.imgrund@tngtech.com

- Theoretical nuclear astrophysics, 2016
- TU Darmstadt
- Business intelligence for other major telco
- Python
- alexander.bartl@tngtech.com

TNG TECHNOLOGY CONSULTING

TNG

Facts and Figures

Founded in 2001	Location Bavaria
Employees 363	Annual Growth Approximately 20%
Employees with a University Degree 99%	Employees with a PhD 60%

TNG Technology Consulting is a value-based consulting partnership focused on high-end information technology.

Six software engineering topics we wish we had known more about while still in academia

Git

Code Structure and Clean Code

Testing

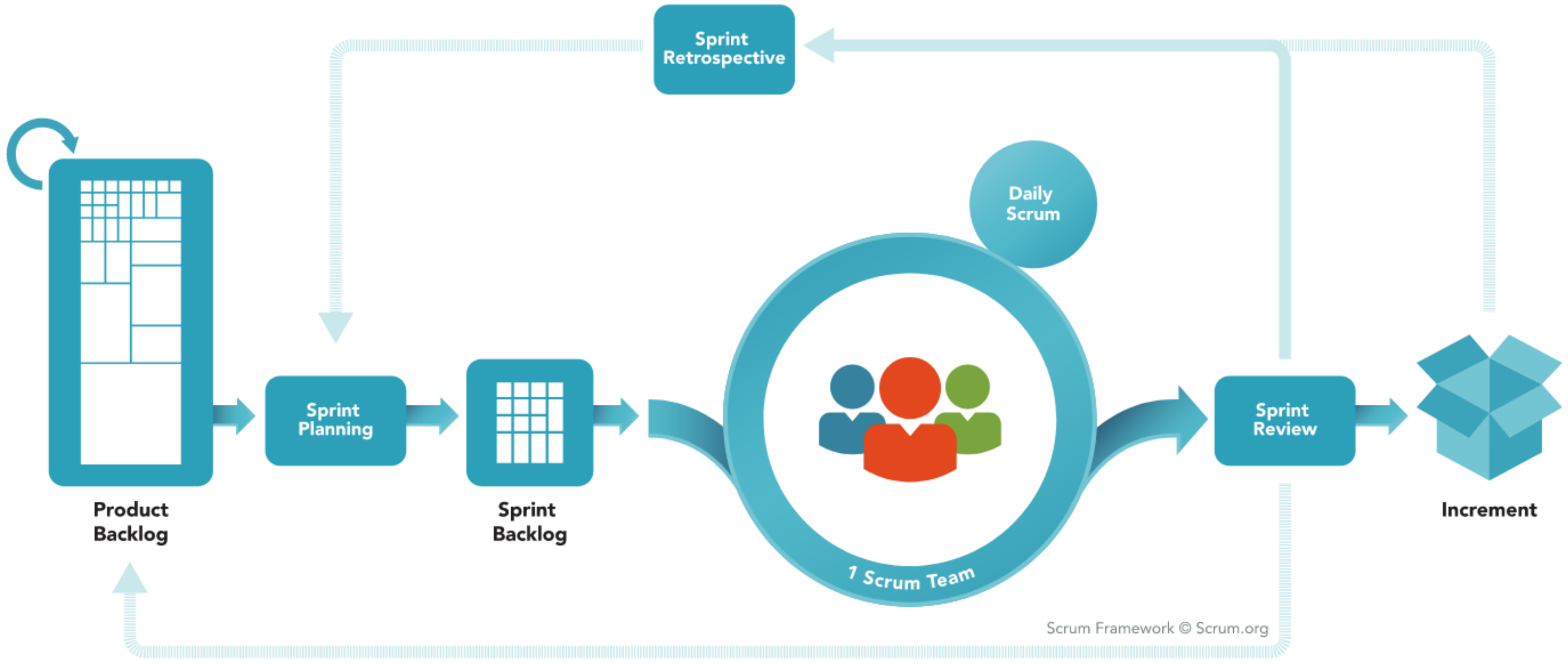
Continuous Integration

Code Reviews

IDEs

Agile Software Development

SCRUM FRAMEWORK



Typical Tooling

Jira Software

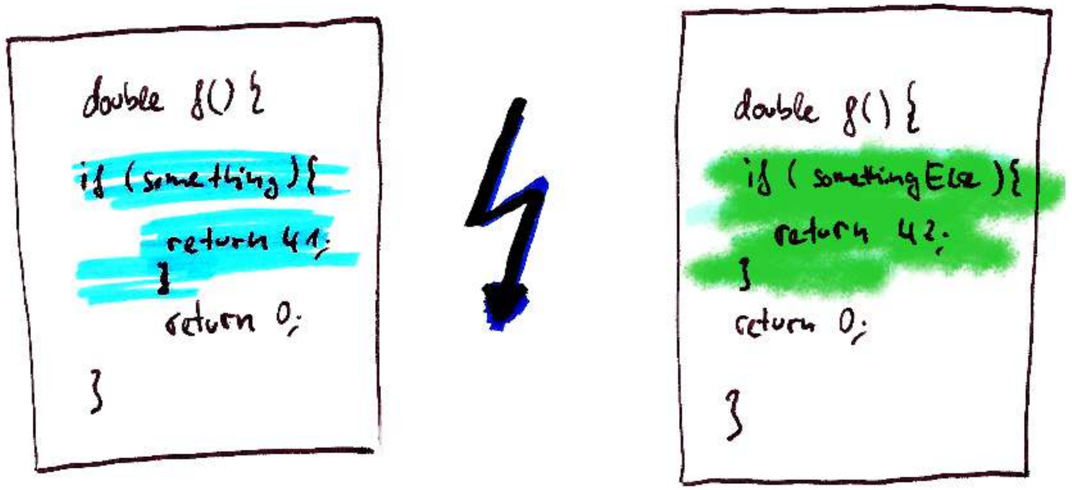
Atlassian Bitbucket

Jenkins

git

docker

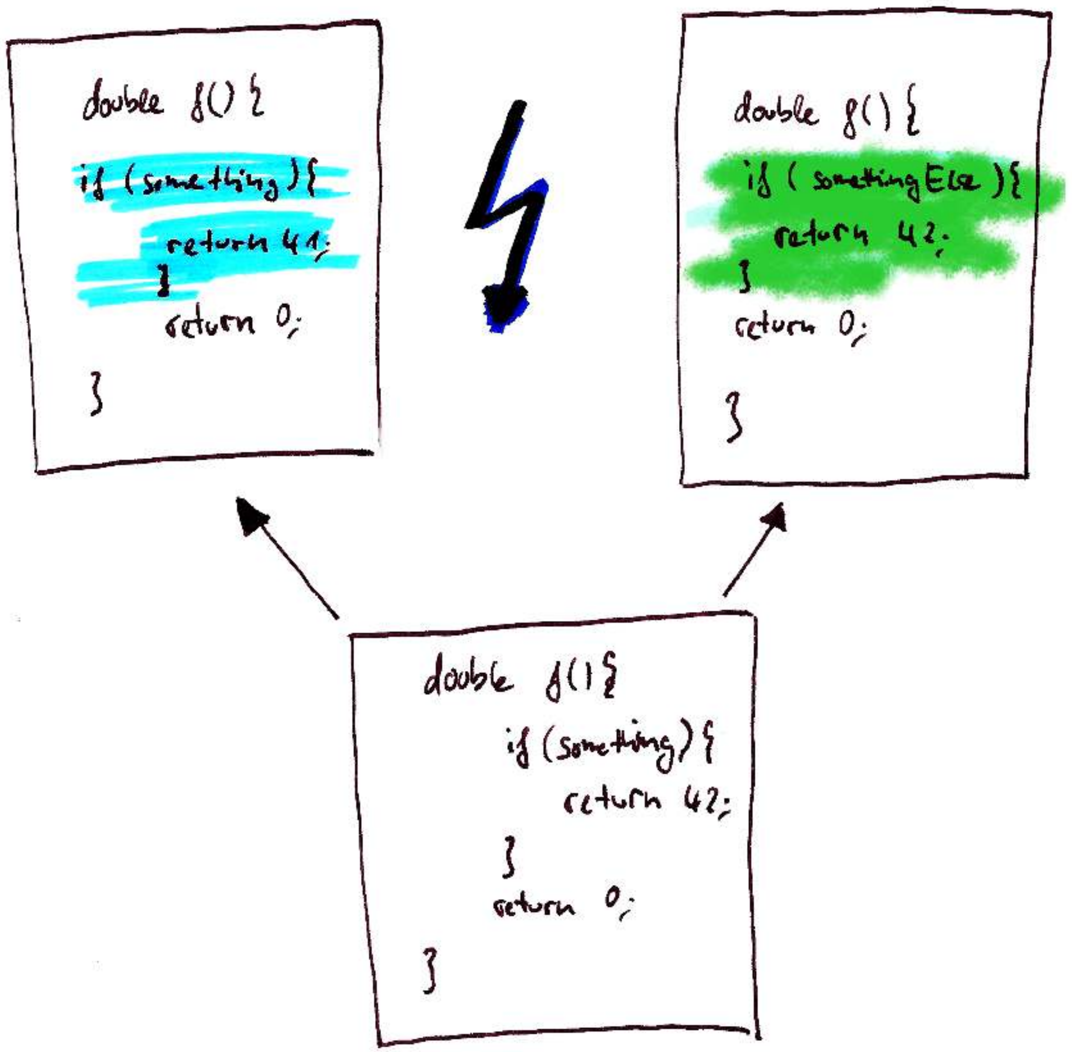
conflicting versions



```
double f() {
  if (something) {
    return 41;
  }
  return 0;
}
```

```
double f() {
  if ( somethingElse ) {
    return 42;
  }
  return 0;
}
```

three-way merge

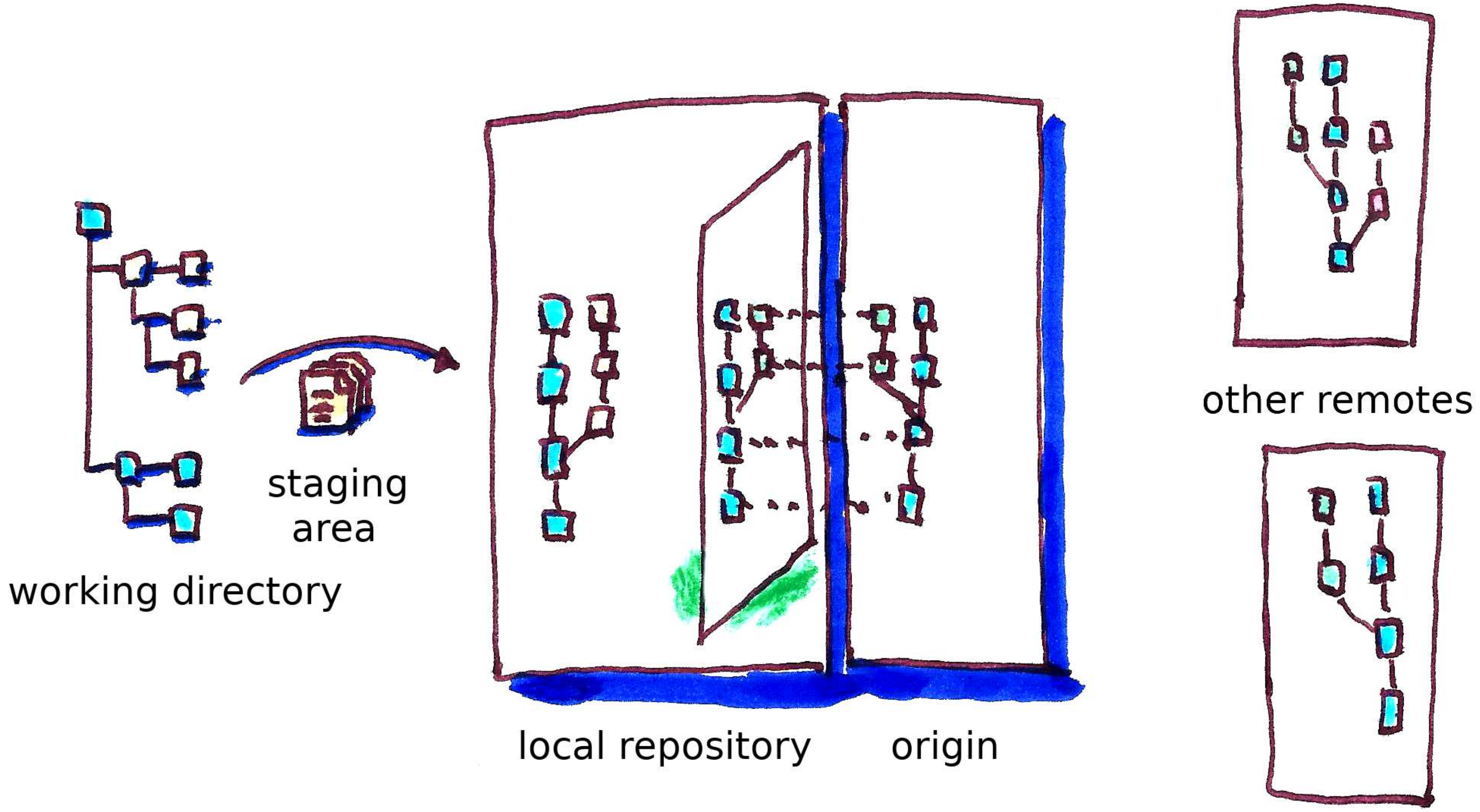


```
double f() {
if (something) {
    return 41;
}
    return 0;
}
```

```
double f() {
if ( somethingElse ) {
    return 42;
}
return 0;
}
```

```
double f() {
        if (something) {
            return 42;
        }
        return 0;
}
```

git architecture



note about commit messages

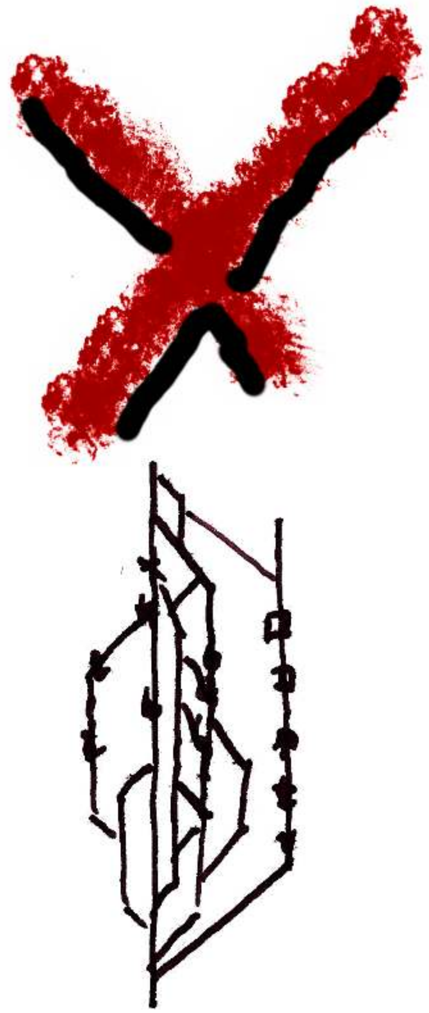
Bad:

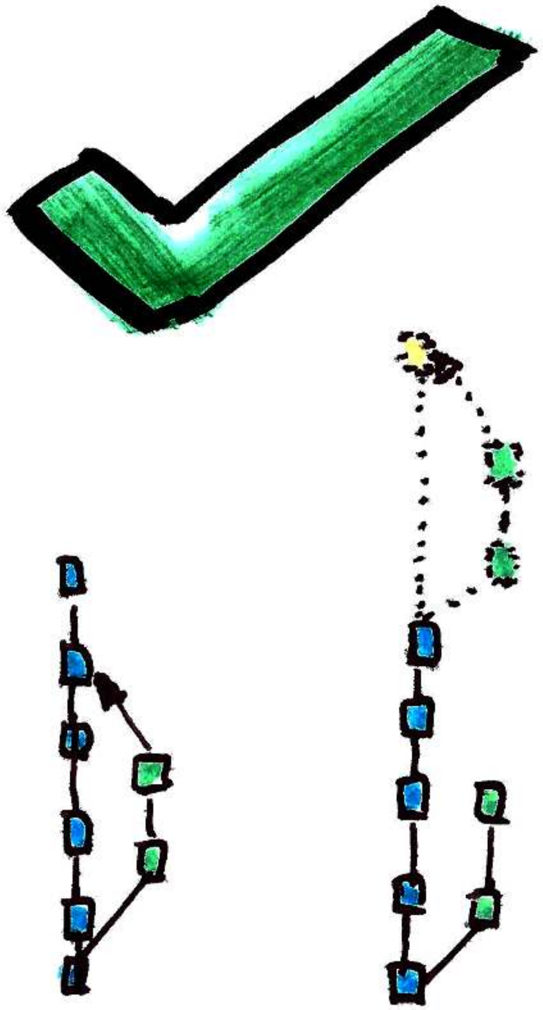
```
make solver convert again

did not converge on my function
```

Good:

```
JIRA-1337 Stabilize matrix solver for special matrices

We implement the algorithm by Holes (DOI:12571.123765)
to ensure convergence when solving the matrix equation.
This was needed to handle matrices with negative
eigenvalues.
```

merge strategies





cool git features

branches

stashing

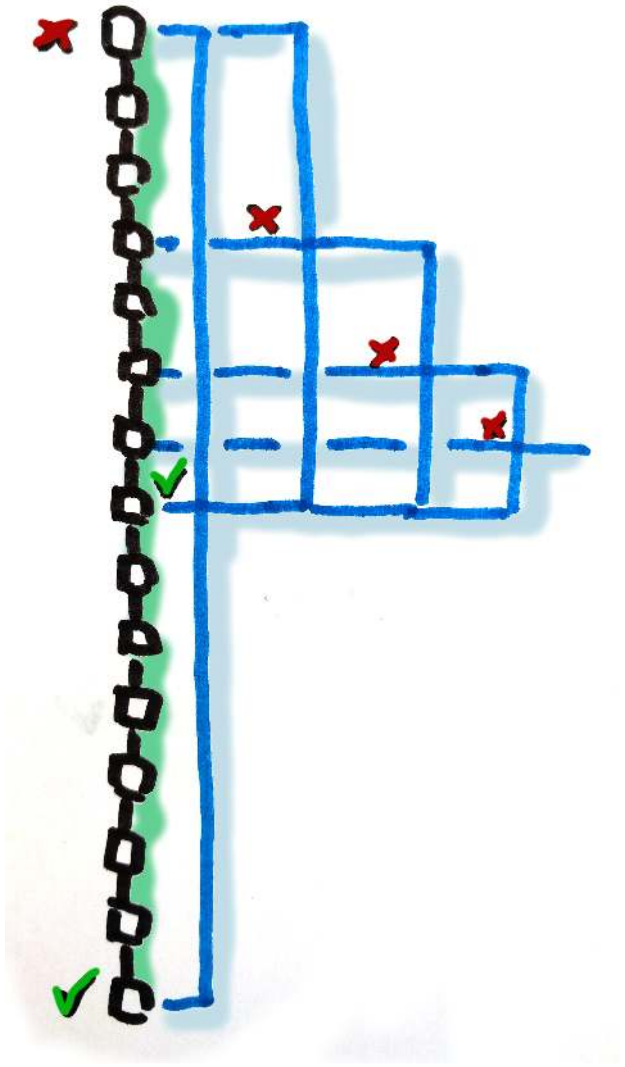
rebase & squash

blame

hooks

bisect

git bisect



only 4 steps to scan 16 commits

- find bugs...
- ...fast
- ...automatically (git bisect run)
- ...visually (git bisect visualize)

Structure and Clean Code

Software Craftsmanship

Short methods

Spaghetti code (comprised of few, long code blocks)

- is hard to comprehend
- can be virtually impossible to test
- hinders code reuse

Instead, methods should only have a few lines of code.

What's in a name?

- consistency: same/similar concepts should have same/similar names
- long names rather than `updIntSec`
- expressiveness of names should reflect the scope
- put *magic numbers* into variables

```
number_of_bins = 5
[init_bin(i) for i in range(number_of_bins)]
```

- Extract conditionals into functions

```
if (time.time() - obj.last_update) > obj.update_interval:
```

```
if needs_update(obj):
```

TNG TECHNOLOGY CONSULTING

The Principle of Least Surprise

A function/method/class should do or hold what one might reasonably expect looking at the name/signature.

Also, there should be no unexpected side effects.

```
def calculate_some_measure():
    os.remove('results/measure/*')
    result = do_some_calculation()
    write_measure_to_disk(result)
    return result
```

Comments

- Good comment:

```
def calculate_some_quantity():
    """ This method implements the algorithm from
        Miller et al, DOI 10.1000/182 """
```

- Bad comment:

```
def do_calc(*args):
    """ calculate the median """
    return np.median(args)
```

TNG TECHNOLOGY CONSULTING

Comments

- Really bad comment:

```
def do_calc(*args):
    """ calculate the median """
    return np.mean(args)
```

- Comments should say things that the code cannot say for itself.
    - References
    - Explanations of why something is done in a specific way
    - Usage info

TNG TECHNOLOGY CONSULTING

DRY, KISS, and the Rule of 3

- **D**on't **R**epeat **Y**ourself
- **K**eep **I**t **S**imple, **S**tupid!
- Rule of Three: If you need some functionality for the third time, properly refactor it into a generalized method
- YAGNI: **Y**ou **A**in't **G**onna **N**eed **I**t

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it? - Brian Kernighan

Design Patterns

- Reusable solutions to common problems in software design
- Formalized best practices
- Provide simple names for complex designs, thus enhancing communication
- Most of the well-known patterns are related to / based on object-oriented programming

Unit Tests

- cover small pieces of code, typically a method/function
- compare the behaviour or result of that method to expectations
- typically run fast, so they can be run often
- do not guarantee code correctness or the absence of bugs
- Scientific Code can seem hard to (unit) test
    - runs slow, does complex work
    - but is built from testable components (existing or extractable)

Why Bother?

- "I'm sure the results looked different yesterday"
- "Let me just make one minor modification here"
- Guarantees that the result of a method does not change
- Only fix a bug once

Example

```
import unittest
from timeit import timeit

from EM500_SRG import ME



class TestEM500SRG(unittest.TestCase):
    def test_some_numbers(self):
        self.assertAlmostEqual(ME(200, 0, 0, 0, 0, 1), -3.274E-6, 9)
        self.assertAlmostEqual(ME(400, 0, 0, 0, 0, 1), 1.133E-6, 9)
        self.assertAlmostEqual(ME(200, 2, 2, 0, 2, 1), -2.822E-7, 10)

    def test_raises_on_invalid_combination(self):
        with self.assertRaises(IOError):
            ME(200, 2, 0, 0, 0, 1)

    def test_subsequent_calls_are_fast(self):
        self.assertLess(
            timeit(setup='from EM500_SRG import ME; ME(200, 0, 0, 0, 0, 1)',
                   stmt='ME(200, 0, 0, 0, 0, 1)', number=10000),
            .2)
```

Some Best Practices

- do not test library code
    - unless the library is not trustworthy
- Test functionality, not implementation
- Test both typical behaviour and corner cases
- Automate the test suite and keep it fast

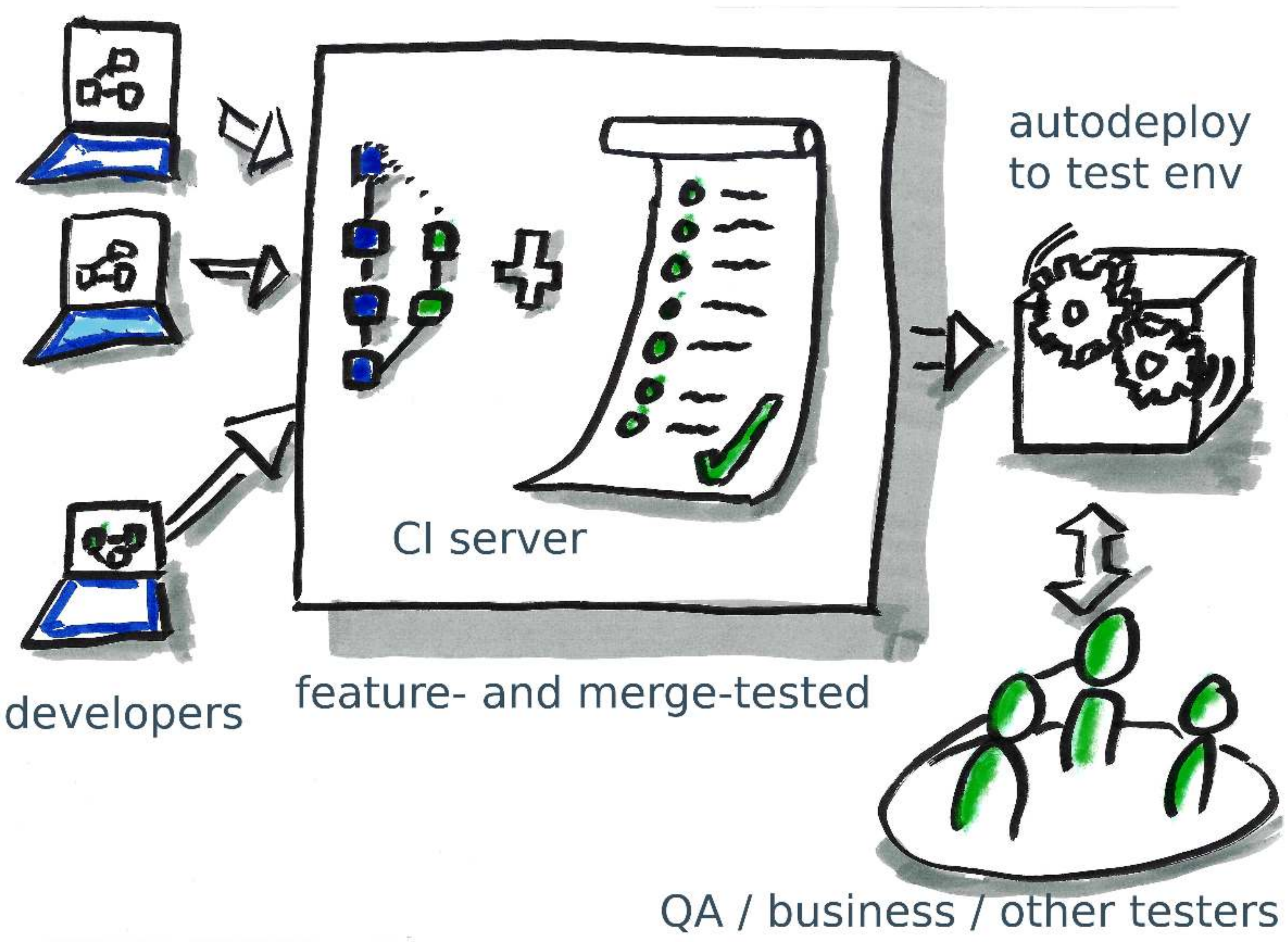
Taking it one step further:

Continuous integration

preconditions:

- common code base
- automated build
- automated self-testing

Continuous Integration



Continuous Integration

May also be useful in science for

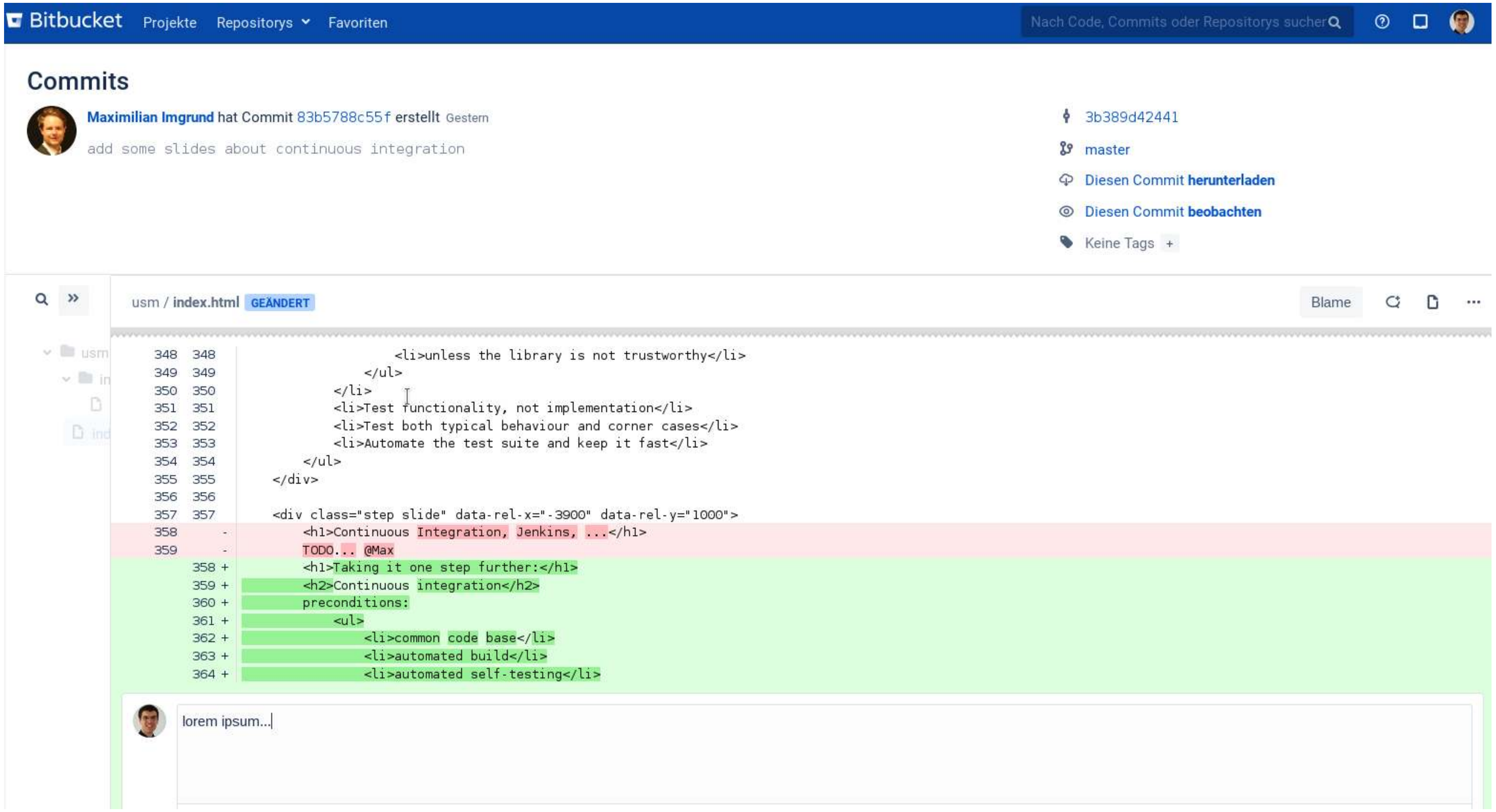
- larger projects with common code base
- continuous monitoring of performance
- regression testing of physics
- regression testing of functionality (MPI-code on cluster?)

Code Reviews

- Read each other's code - repeatedly
- Why?
  - will make it more readable
    → Eagleson's Law of Programming
  - spot errors and performance problems
  - learn from each other
- When? Depends on type of code (shared vs. personal)

TNG TECHNOLOGY CONSULTING

Code Review Example



Commits

Maximilian Imgrund hat Commit 83b5788c55f erstellt Gestern

add some slides about continuous integration

3b389d42441
master
Diesen Commit herunterladen
Diesen Commit beobachten
Keine Tags +

usm / index.html GEÄNDERT

```
                    <li>unless the library is not trustworthy</li>
                </ul>
            </li>
            <li>Test functionality, not implementation</li>
            <li>Test both typical behaviour and corner cases</li>
            <li>Automate the test suite and keep it fast</li>
        </ul>
    </div>

    <div class="step slide" data-rel-x="-3900" data-rel-y="1000">
-       <h1>Continuous Integration, Jenkins, ...</h1>
-       TODO... @Max
+       <h1>Taking it one step further:</h1>
+       <h2>Continuous integration</h2>
+       preconditions:
+           <ul>
+               <li>common code base</li>
+               <li>automated build</li>
+               <li>automated self-testing</li>
```

lorem ipsum...

Live Demo

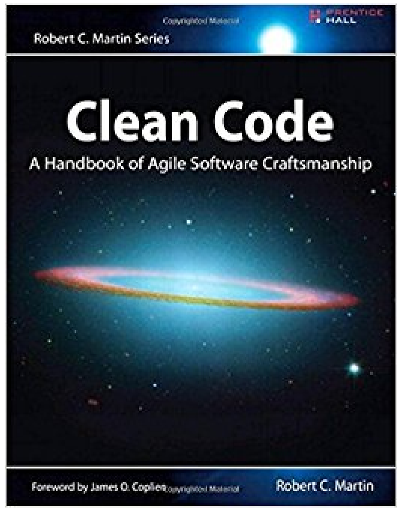
IDEs

IDEs

Facilitate development enormously:

- Saves time
- Saves brain
- Saves nerves
- Eases refactoring

Books

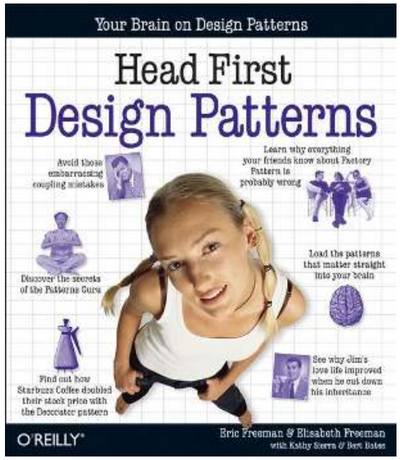


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Robert C. Martin:

Clean Code

Possibly THE book on software craftsmanship, though quite ideological; various summaries publicly available online



Your Brain on Design Patterns
Head First
Design Patterns
O'REILLY
Eric Freeman & Elisabeth Freeman
with Kathy Sierra & Bert Bates

Freeman & Robson:

Head First Design Patterns

Introductory book on design patterns, better than its cover

Resources

- Git

  Learn Git Branching

  Interactive web tutorial to git commands

- Code Structure

  Andrei Boyanov: Python Design Patterns Guide

  Blogpost on how some of these design patterns can be implemented in Python

- Unit Tests

  Gilded Rose Refactoring Kata

  Toy problem for writing unit tests and refactoring tested code, available in many languages

  Python unittest module

  part of the standard library

Resources (2)

- Code Reviews

  Kevin London: Code Review Best Practices

  Blogpost on both the interhuman aspect of blog posts and various topics of clean code

- TNG

  TNG Website and FAQ for prospective applicants

  Come meet us at our Open Techdays for talks, workshops, discussions, and free barbecue on our roof terrace (weather permitting)