

14.–17. 09. 2009
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Simple Code

Regain Control over Software through Incremental Development

Prof. Peter Sommerlad

HSR - Hochschule für Technik Rapperswil
IFS Institut für Software

Peter Sommerlad

peter.sommerlad@hsr.ch



● Work Areas

- Refactoring Tools (C++, Groovy, Ruby, Python) for Eclipse
- **Decremental Development** (make SW 10% its size!)
- Modern Software Engineering
- Patterns
 - Pattern-oriented Software Architecture (POSA)
 - Security Patterns

● Background

- Diplom-Informatiker (Univ. Frankfurt/M)
- Siemens Corporate Research - Munich
- itopia corporate information technology, Zurich (Partner)
- Professor for Software HSR Rapperswil, Head Institute for Software

Credo:

● People create Software

- communication
- feedback
- courage

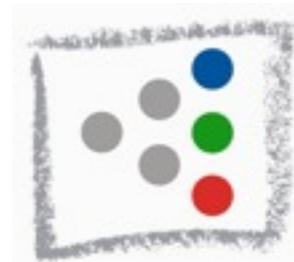
● Experience through Practice

- programming is a trade
- Patterns encapsulate practical experience

● Pragmatic Programming

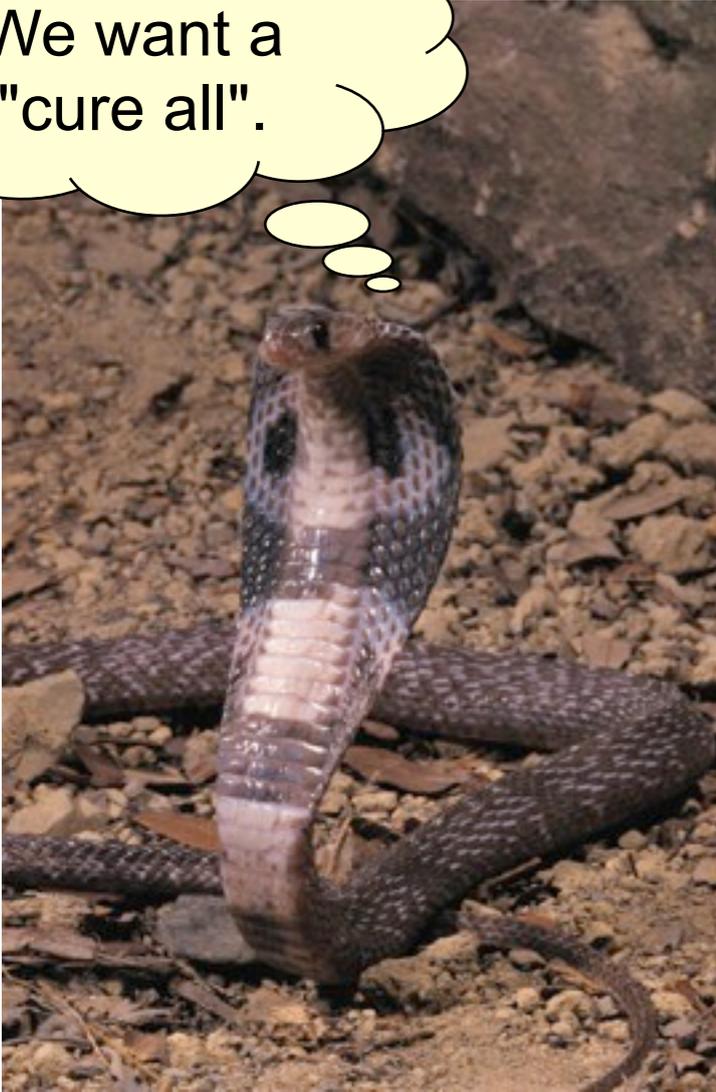
- test-driven development
- automated development
- **Simplicity: fight complexity**

Why we need Decremental Development



- **Problems solved by Software increase**
 - more problems
 - larger & more complex
- **“Good-enough” quality often isn’t**
 - when deployed (Beta-Release)
 - while maintained (updates breaking stuff)
 - with sometimes spectacular failures
- **Useful Software is used longer than intended**
 - pro-active maintenance often neglected
 - repeated bug-hunt-fix-patch deteriorates quality
 - need tools and methods to sustain software

Snake-Oil and Silver Bullets



We want a
"cure all".

- **Acronym Jungle**

- CASE, OOP, CMM, SGML

- more modern:

- XML, EJB, .NET, UML, MDA

- **Technology Overload**

- C++

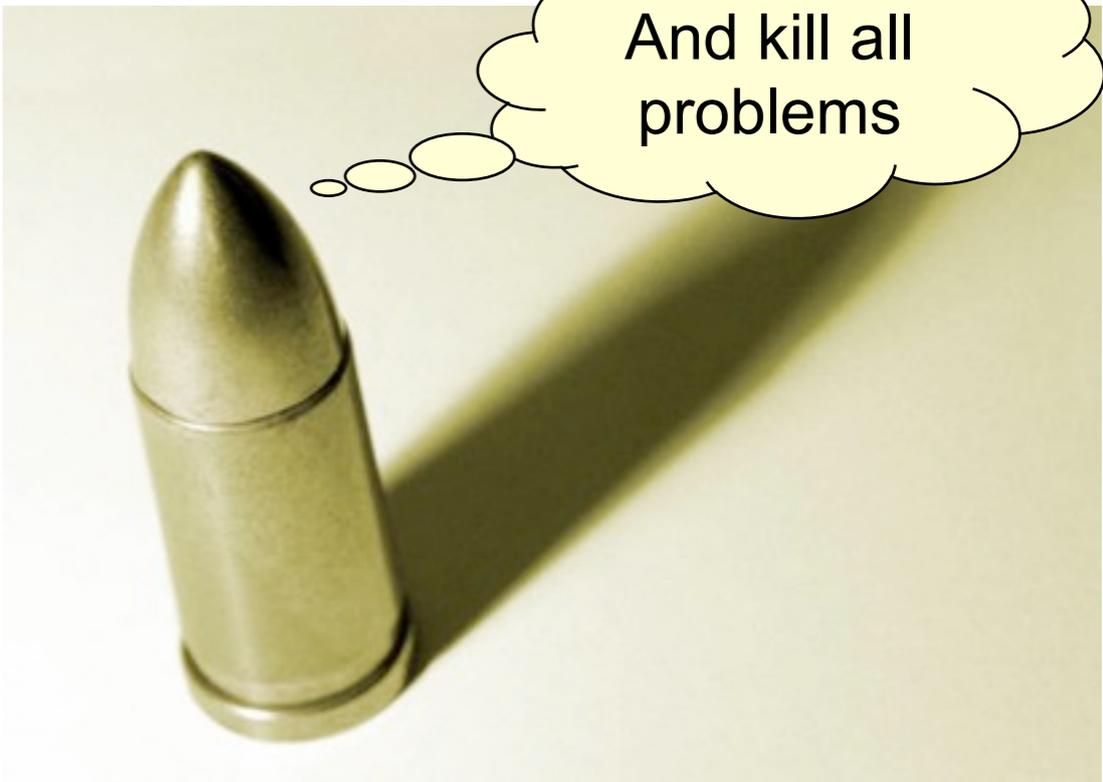
- Corba

- Java

- C#

- VB

- XSLT



And kill all
problems

COMPLEXITY

Complexity is one of the biggest problems with software.

It is much easier to create a complicated "solution" than to really solve a problem.

Much software complexity is accidental not inherent to the problem solved.

Some Reasons for Complexity

- **Young Guns**

- "Hey, I learned so many complicated things, I want to use them now!"
- Coolness is important!
- Complex stuff is cool!
- Over-Engineering

- **"Challenged" Programmers**

- "I don't know how it works, but I made it run."
- Programming by Coincidence
- No idea of Abstraction
- Copy-Paste Reuse
- Under-Engineering

- **Media/Conferences**

- "There is this brand new stuff called XYZ, we tell you how to achieve productivity increase with it"
- sells only "newest" stuff

- **Consultants**

- "We must use XYZ for your problem" ... thinking "because it gives us more billable hours"

- **Resume-oriented Developer**

- "I'll use this cool new stuff, because it looks good on my resume"

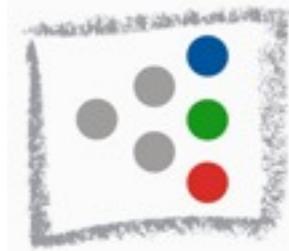
SIMPLICITY

**We need to value Simplicity much higher.
Our software needs to be simpler to solve more
complex problems.
Simple software requires work and skill but
pays off in the long run.**

Famous Quotes by Sir C.A.R.(Tony) Hoare

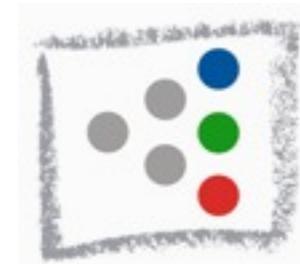


- Inside every large program, there is a small program trying to get out.
- There are two ways of constructing a software design:
 - one way is to make it so simple that there are obviously no deficiencies, and
 - the other way is to make it so complicated that there are no obvious deficiencies.
- The first method is far more difficult.



Less Code
=
More Software

Unnecessary Complexity Starts in the Small



Individuals are guilty!

- **Code Example**

- next slide

- **lack of skill or talent or taste**

- don't know/feel what is "good code"

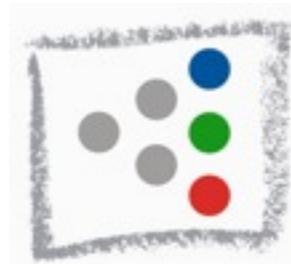
- **thinking "it works"**

- value system not fit for needs

- **copy-waste mentality**

- too much bad code available

Some Examples



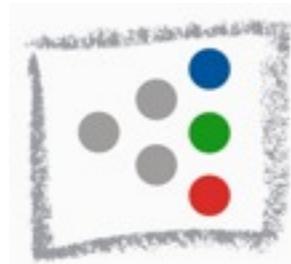
- **ignorance of boolean logic and shortcuts:**

```
if(isOldest)
{
    if(timeLastAccessed < cutOffTime)
    {
        return true;
    }
    else
    {
        return false;
    }
}
else
{
    return false;
}
```

`return isOldest && timeLastAccessed < cutOffTime;`



Programming is too easy: novices learning curve

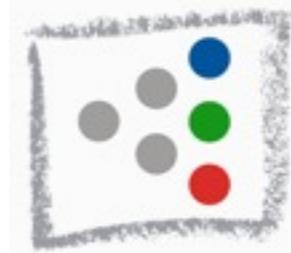


- **“it compiles!”**
 - no syntax error detected by compiler
- **“it runs!”**
 - program can be started
- **“it doesn’t crash”**
 - ... immediately with useful input
- **“it runs even with random input”**
 - the cat jumped on the keyboard
- **“it creates a correct result”**
 - a single use case is working with a single reasonable input

- **Project culture favors delivery of bad software**
 - hard to judge investment in future
 - fire-and-forget orders
 - deployment date fixation
 - manual testing still mainstream
- **“Maintenance” is for janitors**
 - low-profile job
 - no one wants to clean the crap
 - no budget for creating better abstractions/code
 - only adding code or fixing bugs, as little change as possible, because of risk of breaking something
- **lack of feedback, lack of reaction to feedback**



Project Perspective



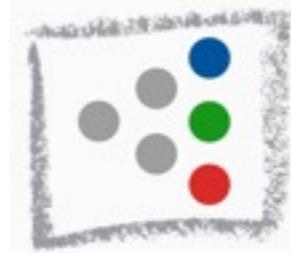
In project management a **project** consists of a **temporary** endeavor undertaken to create a **unique** product, service or result

PMBOK Guide

- **calls for “shortcuts” through defined ending**
 - “we fix it later” means “never”!
 - initial design debt owed
- **useful software lives long** (not only temporary)
 - multiple releases require long term commitment
 - interest to be payed and debt often increases
- **Complexity gets introduced from the beginning**
 - Abstraction seems to only pay-off long term
- **more billable hours with bad code!**

Long term problems

Dirty Code



- **Dirty Code requires clean up**
 - sometimes covering up is insufficient
 - see pictures!
- **Changing Dirty Code is hard**
 - hard to distinguish valuable remains from
 - crap
 - Cleaning requires change
- **Release cycles get longer and longer**
 - with fewer and or more buggy features
 - can bring down companies or departments
 - worst case: brings down customer!
 - “We will redesign later” (**means never**)

SMDK
Kölliken

Decremental Development

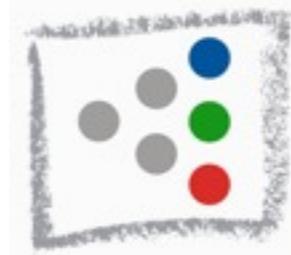


- **Reduce software size TO 10%**

- while keeping required functionality
- while improving its quality
- while improving its design

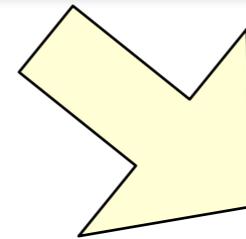
➤ measure productivity by Lines of Code (LoC) removed

One means of reduction -> choice of tool



```
using System;

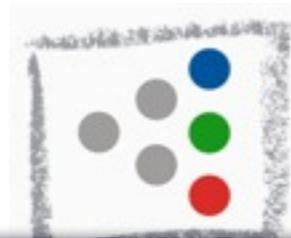
class HelloWorld
{
    public static int Main(String[] args)
    {
        Console.WriteLine("Hello, World!");
        return 0;
    }
}
```



Just think of XML if you need to
think of complexity in syntax

puts "Hello, World!"

Hiding + Knowledge



```
a=[42, 1, 7, 2, 34, 64, 29, 2]
```

```
for i in 0..a.length-1 do  
  for j in i+1...a.length do  
    if (a[i] > a[j]) then
```

```
      x = a[i]  
      a[i] = a[j]  
      a[j] = x
```

```
    end
```

```
  end
```

```
end
```

```
puts "a:"
```

```
puts a
```

```
de c
```

```
e
```

```
er
```

```
for
```

```
if
```

```
end
```

```
end
```

```
end
```

```
end
```

```
for i in 0..a.length-1 do  
  for j in i+1...a.length do  
    if (a[i] > a[j]) then  
      a[i], a[j] = a[j], a[i]
```

```
    end
```

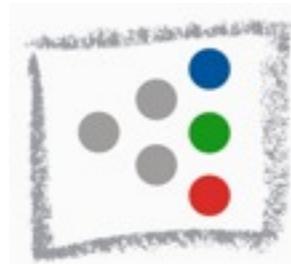
```
  end
```

```
end
```

```
end
```

```
end
```

Use existing stuff from libraries -> Knowledge



```
a=[42, 1, 7, 2, 34, 64, 29, 2]
for i in 0..a.length-1 do
  for j in i+1...a.length do
    if (a[i] > a[j]) then
      a[i], a[j] = a[j], a[i]
    end
  end
end
puts "a:"
puts a
```

```
a=[42, 1, 7, 2, 34, 64, 29, 2]
puts "a:"
puts a.sort
```

Developer Means for Decremental Development



- **Refactoring**

- requires (test-) automation
- higher-level restructuring still missing in tools
 - yet to be implemented (TBI), ideas but no tools yet

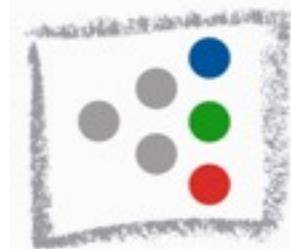
- **Code generation**

- as simple as possible, for getting “DRY” code
 - shouldn't require complex XML :-)

- **Code-smell detectors (new ideas, TBI)**

- lint, FindBugs, metrics etc. are only the beginning
- even better: Design-odor detectors
- with automatic deodorant application :-)





- **Crisis might allow for change in Value System**
 - short term profit no longer fashionable, when it means increased risk
- **Simplicity vs. Techno-hype**
 - hard to establish, doesn't mean not using hi-tech
- **Learning and valuation of skill**
 - no 1000 monkeys with a keyboard
- **Maintenance by best skilled people**
 - who would you hand a valued painting for restoration?
- **Leadership by software experts**
 - not only "management of resources"

Research Proposal

Decremental Development



- **Create better tools for automated Refactoring**
 - for languages lacking support, but with large code bases, i.e.,
 - C/C++, Java, PL/I, Ada, COBOL(?)
 - Groovy, Ruby, javascript, PHP, ...
- **Develop new approaches for higher-level software simplification**
 - beyond Refactoring
 - i.e., detecting potential for simplification
- **Increase valuation of Simplicity**
 - as a software design goal
 - articles, presentations, case studies, talks

How do we get there?

or: A Way to Simple Code



- **What can YOU do today or tomorrow?**
 - Value Simplicity and Simple Code
 - Establish and improve developer skills
 - Clean Code attitude
 - SOLID design principles
 - Continuous design improvement
 - Refactoring
 - Test automation with GUTs
 - Software lifetime perspective beyond initial project
 - join in at <http://wiki.hsr.ch/SimpleCode>
- **Start NOW!**

Questions?



- join in at <http://wiki.hsr.ch/SimpleCode>
- or contact me at **peter.sommerlad@hsr.ch**

14.–17. 09. 2009
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Prof. Peter Sommerlad

HSR - Hochschule für Technik Rapperswil
IFS Institut für Software