Simple Code
Regain Control over Software through Decremental Development

Prof. Peter Sommerlad
HSR - Hochschule für Technik Rapperswil
IFS Institute for Software
Oberseestraße 10, CH-8640 Rapperswil
peter.sommerlad@hsr.ch
http://ifs.hsr.ch
http://wiki.hsr.ch/PeterSommerlad
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

Acronym Jungle
- CASE, OOP, CMM, SGML
  - more modern:
    - XML, EJB, .NET, UML, MDA

Technology Overload
- C++
- Corba
- Java
- C#
- VB
- XSLT

We want a "cure all". And kill all problems.
Complexity is one of the biggest problems with software if not THE biggest. 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"
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.
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
Unnecessary Complexity Starts in the Small

- ignorance of boolean logic and shortcuts:

```java
if(isOldest)
{
  if(timeLastAccessed< cutOffTime)
  {
    return true;
  }
  else
  {
    return false;
  }
}
else
{
  return false;
}
```
Unnecessary Complexity Starts in the Small

- ignorance of boolean logic and shortcuts:

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

```java
return isOldest && timeLastAccessed < cutOffTime;
```
Programming is too easy: novices learning curve

• “it compiles!”
  o no syntax error detected by compiler

• “it runs!”
  o program can be started

• “it doesn’t crash”
  o ... immediately with useful input

• “it runs even with random input”
  o the cat jumped on the keyboard

• “it creates a correct result”
  o a single use case is working with a single reasonable input
Cultural Reasons

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

- **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, design 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

removal of highly toxic waste at SMDK Kölliiken, CH
Long term problems

Dirty Code

- Dirty Code requires clean up
  - sometimes covering up is insufficient
    - see pictures!

removal of highly toxic waste at SMDK Kölliken, CH
Long term problems
Dirty Code

● Dirty Code requires clean up
  o sometimes covering up is insufficient
    ➢ see pictures!

● Changing Dirty Code is hard
  o hard to distinguish valuable remains from
    ➢ crap
  o Cleaning requires change

removal of highly toxic waste at SMDK Kölliiken, CH
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)

removal of highly toxic waste at SMDK Kölliken, CH
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 removed (LoCR)
One means of reduction
-> choice of tool

```csharp
using System;

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

puts "Hello, World!"
One means of reduction
-> choice of tool

```csharp
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
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
Hiding + Knowledge

```ruby
# 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
```

```ruby
def swap a, i, j
  x = a[i]
  a[i] = a[j]
  a[j] = x
end

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

Hiding + Knowledge

```ruby
def swap a, i, j
  a[i], a[j] = a[j], a[i]
end
for i in 0..a.length-1 do
  for j in i+1...a.length do
    if (a[i] > a[j]) then
      swap(a, i, j)
    end
  end
end
puts "a:
puts  a
```

---

```ruby
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
```

---

```ruby
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
```

---

```ruby
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
      swap(a, i, j)
    end
  end
end
puts "a:
puts  a
```
Hiding + Knowledge

```ruby
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
```

ACCU 2011 Simple Code © Prof. Peter Sommerlad
Use existing stuff from libraries
-> Knowledge

```ruby
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
```
Use existing stuff from libraries

-> Knowledge

```ruby
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
```
Use existing stuff from libraries

-> Knowledge

```ruby
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**
  - lint, FindBugs, metrics etc. are only the beginning
    - see for example: linticator.com and includator.com
  - even better: Design-odor detectors
  - with automatic deodorant application :-(
Cultural Means

- **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? the brand new apprentice?

- **Leadership by software experts**
  - not only “management of resources”
What we are doing at IFS
Decremental Development

- Create better tools for automated Refactoring
  - for languages lacking support, but with large code bases, i.e.,
    - C++, (COBOL, PL/I, Ada, ...)?
    - Scala, Groovy, Ruby, Python, Javascript, PHP, ...

- Develop new approaches for higher-level software simplification
  - beyond Refactoring: lint, generators, ...
  - i.e., detecting potential for simplification

- Increase valuation of Simplicity
  - as a software design goal
  - articles, presentations, case studies, talks
How do we all get there?

- **What can YOU do today or tomorrow?**
  - Value Simplicity and Simple Code
  - Establish and improve developer skills
  - Clean Code attitude (see Bob Martin's book)
  - SOLID design principles
  - Continuous design improvement
    - Refactoring
  - Test automation with GUTs
  - Software lifetime perspective beyond initial project

- **Start NOW!**
Questions?

- or contact me at peter.sommerlad@hsr.ch