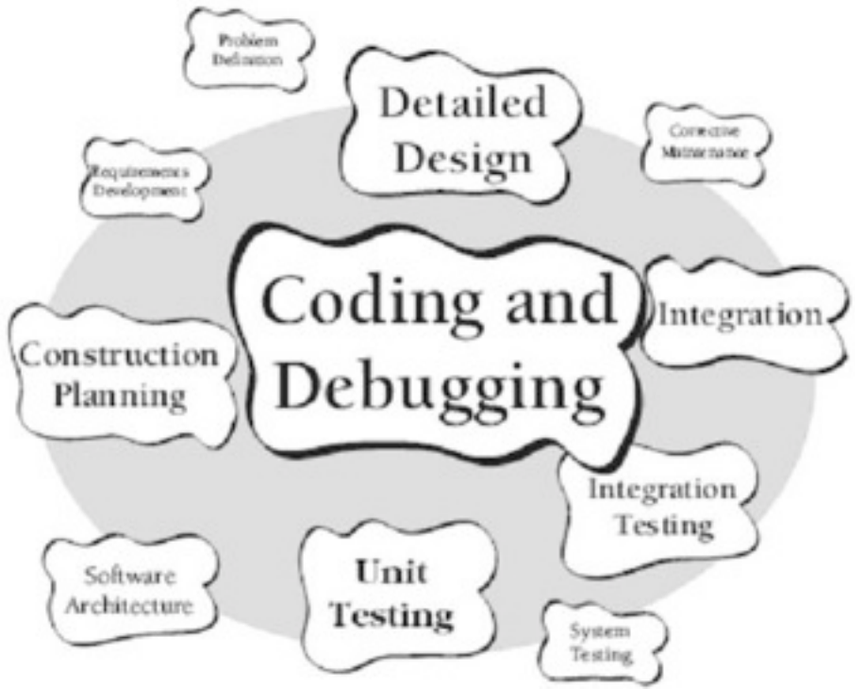


Software Construction

2011-2012
Tijs van der Storm (storm@cwi.nl)

UNIVERSITEIT VAN AMSTERDAM

CWI

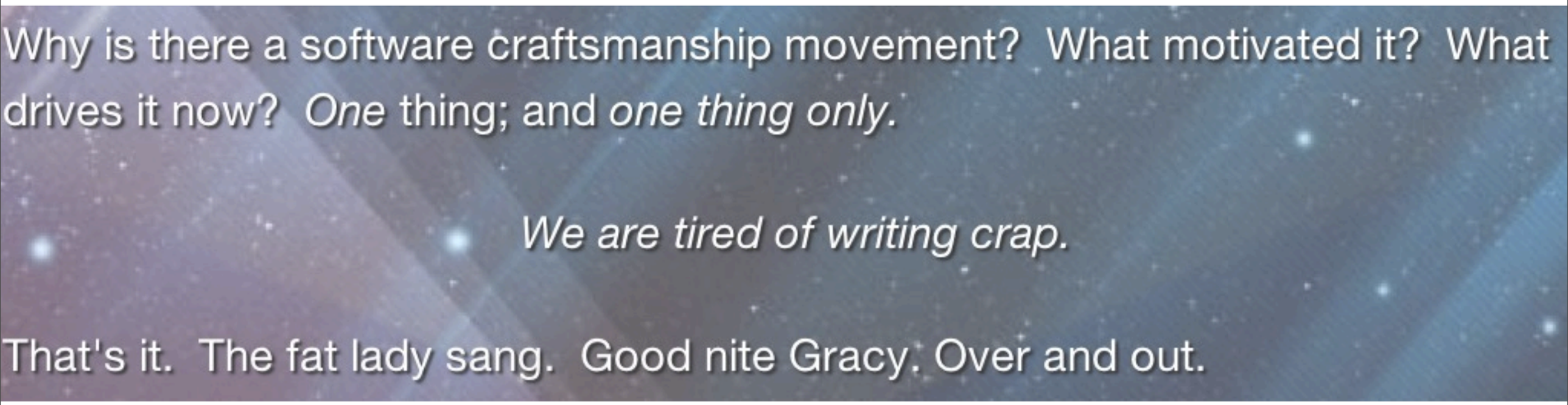
What this course is about

- You all know programming, right?
- But what is good code?
- How to *reason* about good code?
- Think about it.

This course is *not* about

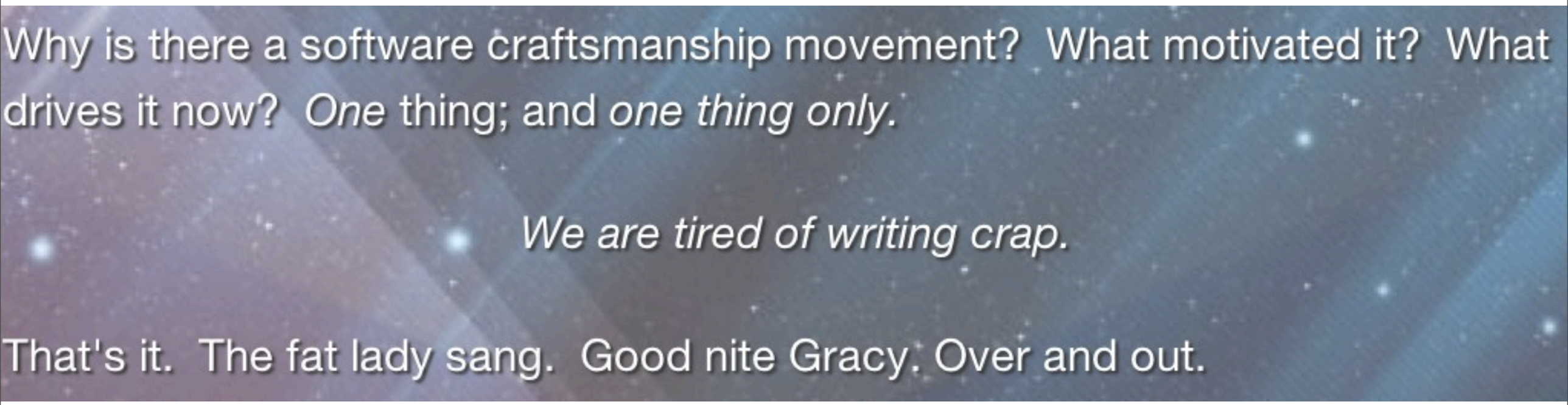
- Data structures
- Algorithms
- Programming language X
- Paradigm X (though: OO)
- GUI programming
- Web applications
- Concurrency
- Performance
- Graphics programming
- Mathematics
- Computational complexity
- ...

Uncle Bob*



*Robert Martin, http://cleancoder.posterous.com/software-craftsmanship-things-wars-commandmen

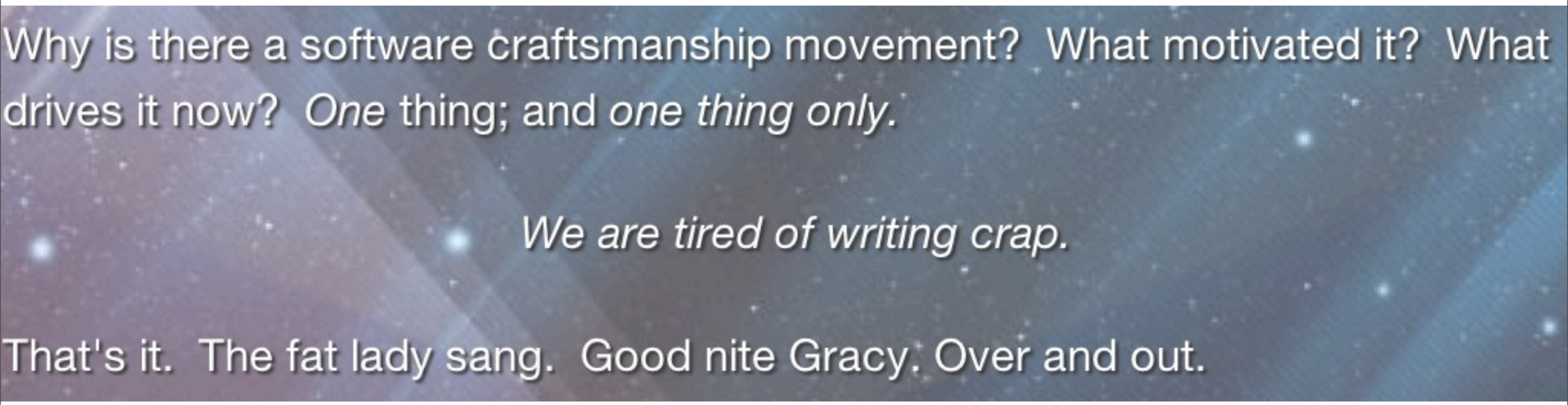
Uncle Bob*



This course is *not* about the software craftmenship movement...

*Robert Martin, http://cleancoder.posterous.com/software-craftsmanship-things-wars-commandmen

Uncle Bob*

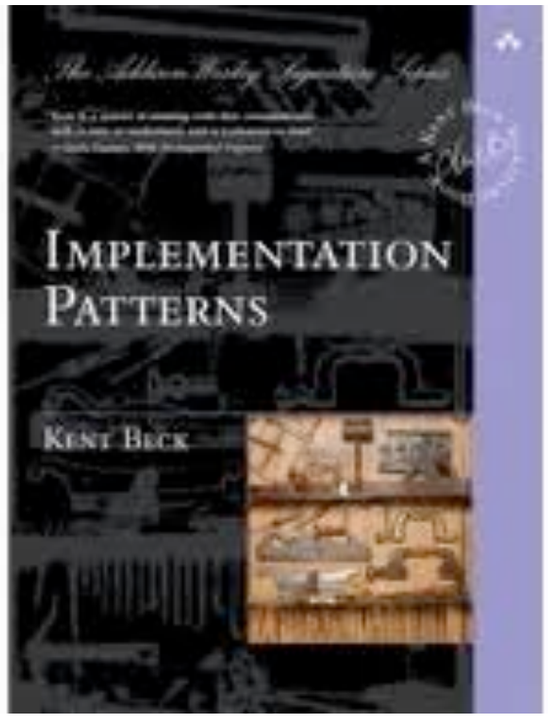


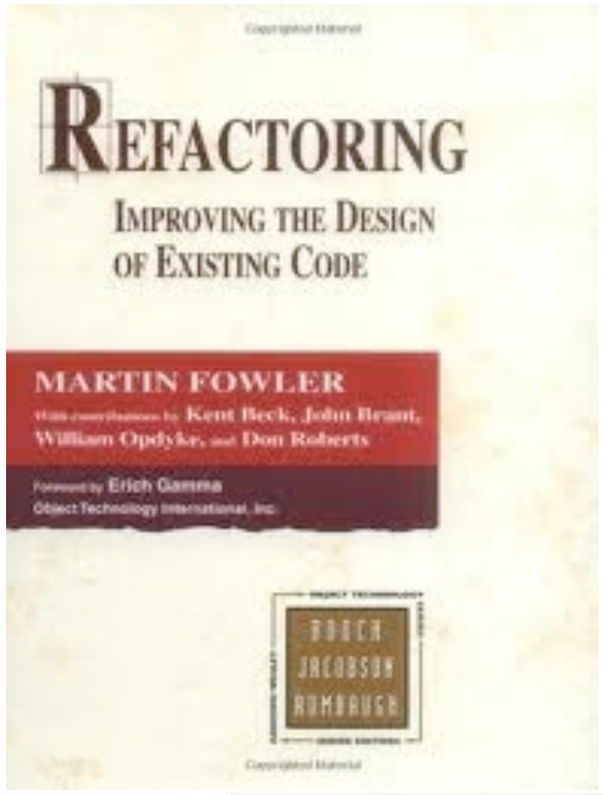
This course is *not* about the software craftmenship movement...

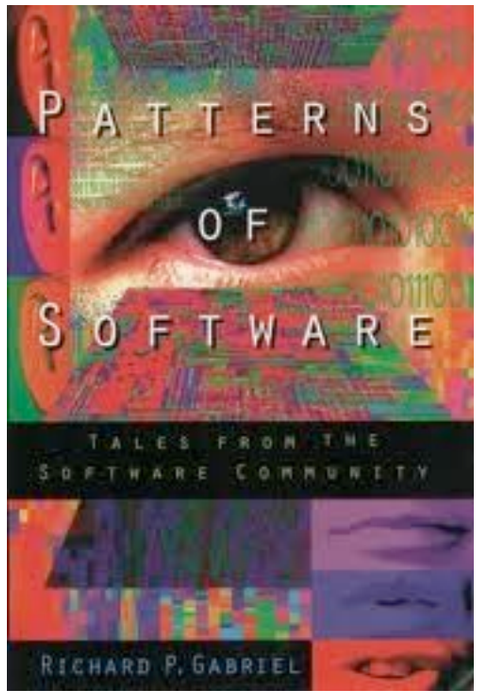
This course is about *not* writing crap.

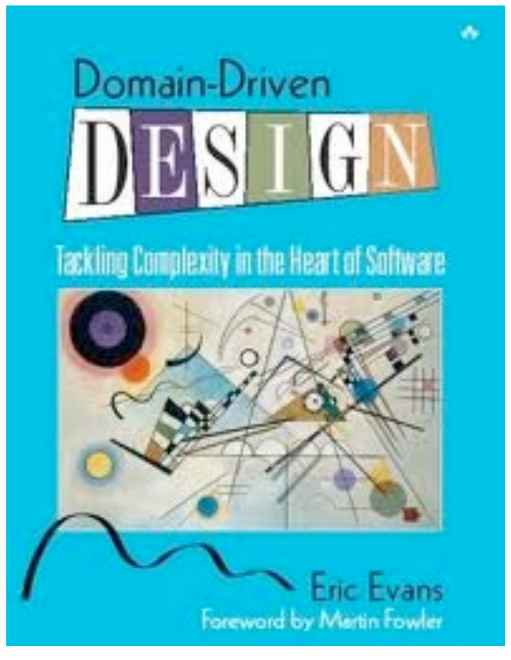
*Robert Martin, http://cleancoder.posterous.com/software-craftsmanship-things-wars-commandmen

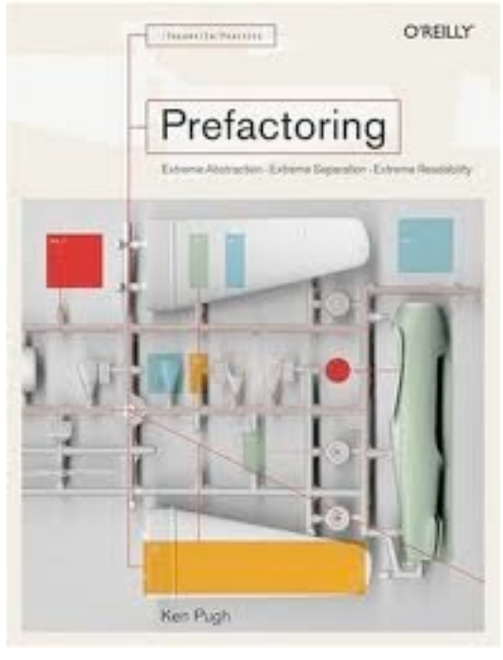
Representative books

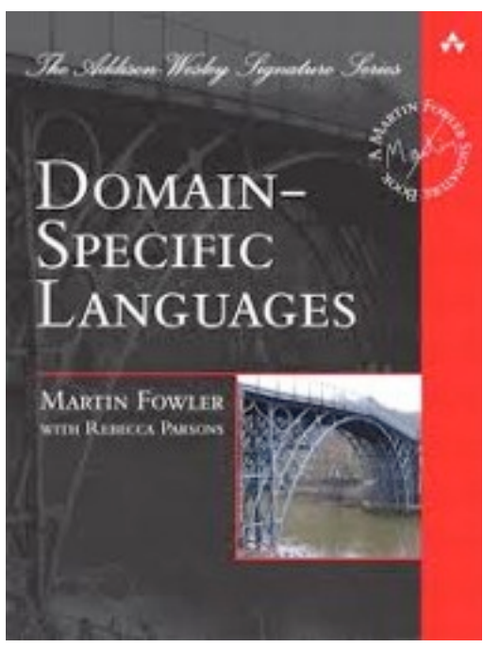












Learning goals

- Create good low level designs
- Produce clean, readable code
- Reflect upon techniques, patterns, guidelines etc.
- Assess the quality of code
- Apply state of the art software construction tools

Overview

- Lectures
- Theory
- Research paper
- Lab assignment
- Concluding

Lectures

Lectures

- Introduction (today)
- Syntax analysis
- Domain-specific languages
- Code quality
- Debugging

Guest lectures

- Jeroen van den Bos: DSL for digital forensics
- Eelco Visser: Linguistic abstraction for the Web

“Theory”

Two tests

- Online or on paper (not sure yet)
- No grade, but you must *pass* them
- “Immediate” grading

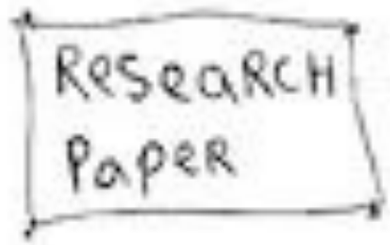
1st test: technique

- Bertrand Meyer, *Applying "Design by Contract"*, 1992, Meyer92.
- Karl J. Lieberherr, Ian M. Holland, *Assuring Good Style for Object-Oriented Programs*, 1989,LieberherrHolland89.
- Robert C. Martin, *The Open-Closed Principle*, 1996, Martin96.
- Ralph Johnson, Brian Foote, *Designing reusable classes*, 1988, JohnsonFoote88.
- Marjan Mernik et al. *When and How to Develop Domain Specific Languages*, 2005,MernikEtAl05.

2 test: philosophy

- D. L. Parnas, *On the criteria to be used in decomposing systems into modules* , 1972, Parnas72
- William R. Cook, *On understanding data abstraction, revisited*, 2009, Cook09.
- Herbert Simon, *The Architecture of Complexity*, Simon62
- Carliss Y. Baldwin, Kim B. Clark, *Modularity in the Design of Complex Engineering Systems*,BaldwinClark06
- Horst Rittel, Melvin Webber, *Dilemmas in a General Theory of Planning*, RittelWebber84.

Research paper



Exercise in argumentation

- Paper: 3-5 pages
- Topics from predefined list
- Choose a *position*
- Argue for/against it
- Required: find 2 more papers in support of your position
- Must be clear you've read all 4 papers

Structured programming with or without gotos?

- E.W. Dijkstra *Goto considered harmful*, 1968, Dijkstra68;
- D. Knuth, *Structured Programming with go to statements*, 1974, Knuth74.

State Transactional Memory

- Simon Peyton Jones, *Beautiful Concurrency*, 2007, PeytonJones07.
- Calin Cascaval et al. *Software Transactional Memory: Why is it Only a Research Toy?*, 2008,CascavalEtAl08.
- Bryan Cantrill and Jeff Bonwick, *Real-world Concurrency*, 2008, CantrillBonwick08.

Internal vs external DSLs

- Marjan Mernik et al. *When and How to Develop Domain Specific Languages*, 2005,MernikEtAl05.
- Martin Fowler, *Implementing an Internal DSL*, 2007 Fowler07.

Aspect-Oriented Programming

- Gregor Kiczales et al. *Aspect-Oriented Programming*, 1997, KiczalesEtAl97.
- Robert E. Filman, Daniel P. Friedman, *Aspect-Oriented Programming is Quantification and Obliviousness*, 2000, FilmanFriedman00.

Literate Programming in the 21st Century

- Bentley, Knuth and McIllroy, *A Literate Program*, 1986, BentleyEtAl86.
- Knuth, *Literate Programming*, 1984, Knuth84.

Prototype-based vs class-based object-orientation

- James Noble, Brian Foote, *Attack of the Clones*, 2002, NobleFoote02.
- Henry Lieberman, *Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems*, 1986, Lieberman86.

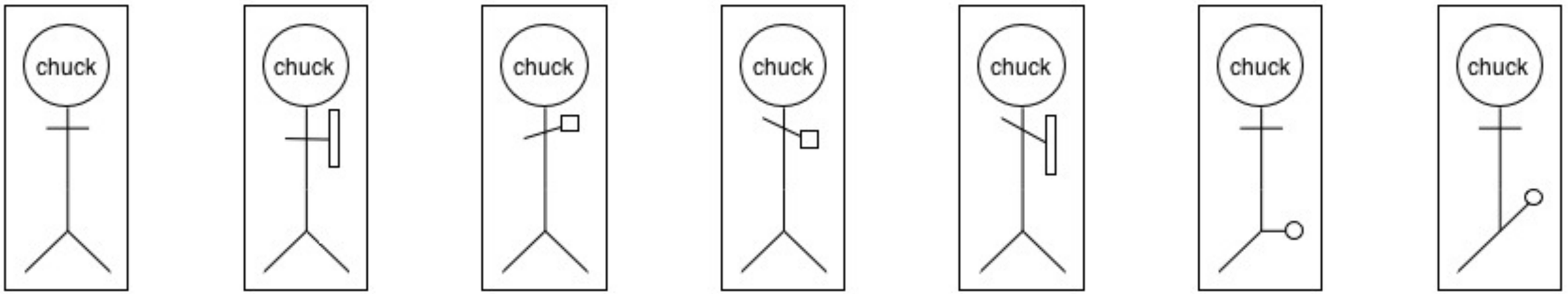
Design by Contract

- Bertrand Meyer, *Applying "Design by Contract"*, 1992, Meyer92.
- Jean-Marc Jézéquel, Bertrand Meyer, *Design by Constract: The Lessons of Ariane*, 1997,JezequelMeyer97.

Fluent or Law-of-Demeter?

- Martin Fowler, *FluentInterface*, 2005, Fowler05.
- Karl J. Lieberherr, Ian M. Holland, *Assuring Good Style for Object-Oriented Programs*, 1989,LieberherrHolland89.

Lab assignment



Lab assignment

Super Awesome Fighters (SAF)

- A DSL for specifying fighter bots
- You will implement this language

```
chicken
{
  kickReach  = 9
  punchReach = 1
  kickPower  = 2
  punchPower = 2
  far [run_towards kick_low]
  near [run_away kick_low]
  near [crouch punch_low]
}
```

Part 1: front end

- Parser: text to abstract syntax tree (AST)
- AST hierarchy
- Type checker
- Two variants:
  - Java
  - Rascal (but: you have to do a little more)

Java

- Select suitable parse generator: *Rats!*, JavaCup, JavaCC, Jacc, ANTLR etc.
- Design AST class hierarchy (required!)
- Visitor/interpreter for checking well-formedness

Rascal

- Rascal is *designed* for making DSLs:
  - syntax definition
  - AST data types
  - type checking
  - desugaring
  - IDE features
- You have to do all of this.

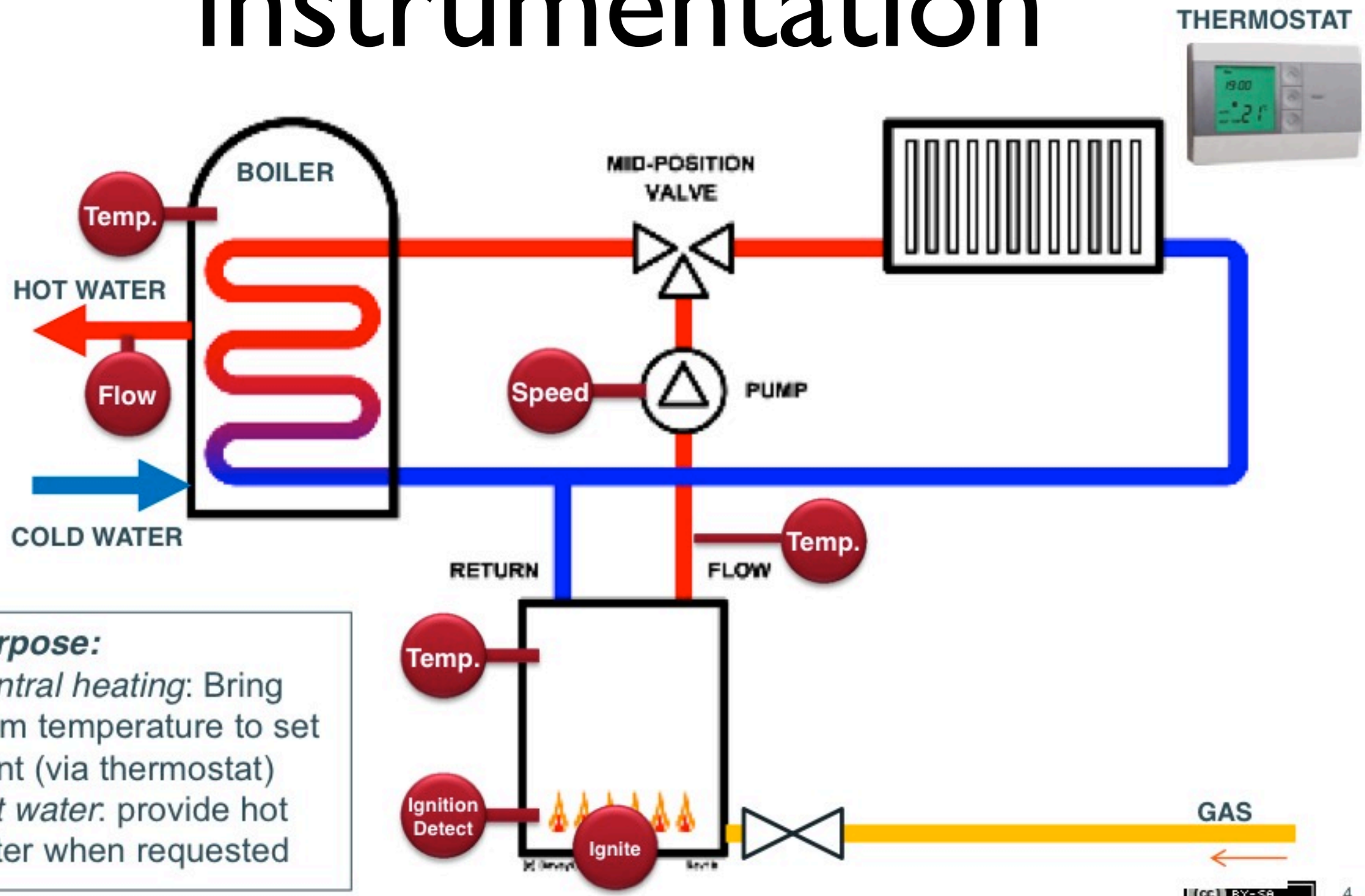
Part 2: Back end

- Implement a game semantics
- Graphical simulation of fighting games
- This part is realized in Java
  - (interface from Rascal through XML)
- *See course info for more detail*
- Bonus: make a compiler

Honors track

- For excellent students
- Implement the Language Workbench Competition 2012 assignment
- ... in Rascal
- Close collaboration with CWI (= me)
- *See course info for details*

DSL for piping and instrumentation

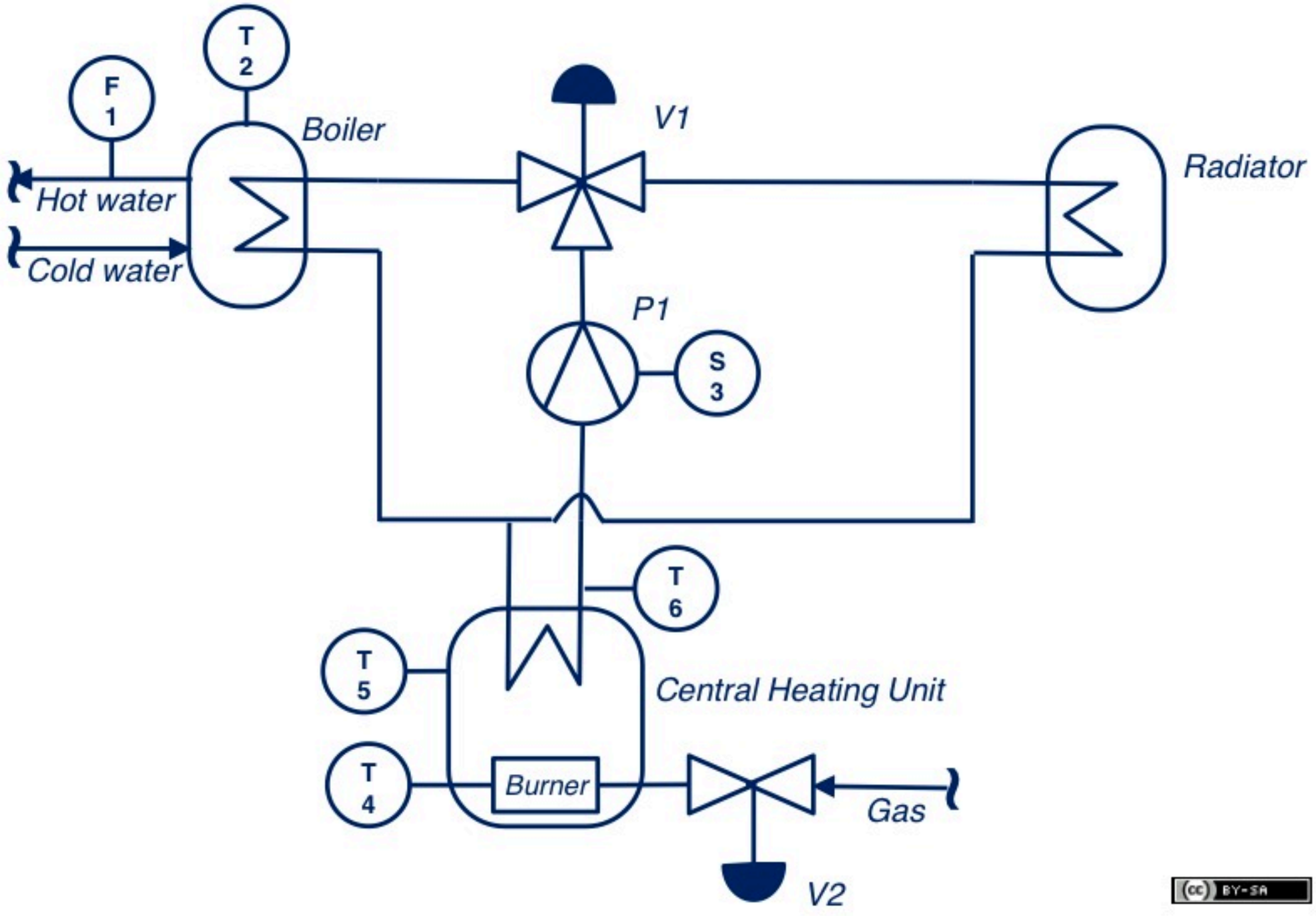


Purpose:
Central heating: Bring room temperature to set point (via thermostat)
Hot water: provide hot water when requested

Two DSLs

- Describing piping and instrumentation models
- A controller language to specify how to control the behavior

Modeling language



Controller language (mock up)

```
MIN_GAS = 50
MAX_GAS = 150
BURNER_RAMPUP = 5
VALVE_SWITCH = 5

state OFF:
 if Radiator.temperature < DESIRED_RADIATOR_TEMP - RADIATOR_MARGIN goto IGNIT
 if Boiler.temperature < DESIRED_WATER_TEMP - WATER_MARGIN goto IGNITE

state IGNITE:
 if Burner.ignite goto RAMPUP
 Burner.ignite = true
 gas = MIN_GAS
 Burner.gas_level = gas
 Pump.run = true

state RAMPUP:
 if Radiator.temperature > DESIRED_RADIATOR_TEMP
    and Boiler.temperature < DESIRED_WATER_TEMP - WATER_MARGIN goto BOILER
 if Radiator.temperature < DESIRED_RADIATOR_TEMP - RADIATOR_MARGIN
    and Boiler.temperature > DESIRED_WATER_TEMP goto RADIATOR
 if Radiator.temperature > DESIRED_RADIATOR_TEMP
    and Boiler.temperature > DESIRED_WATER_TEMP goto RUNNING
 Burner.ignite = true
 gas = gas + BURNER_RAMPUP
 Burner.gas_level = gas
 Pump.run = true

state BOILER:
 if Radiator.temperature < DESIRED_RADIATOR_TEMP - RADIATOR_MARGIN
    goto RAMPUP
```

Required deliverables

- Syntax, checker, IDE features etc. of both DSLs
- Visualization of the Piping models
- Simulation of Piping and/or controller specifications
- Possibly: code generator
- See http://www.languageworkbenches.net/

Subversion

- Assignment to be completed *individually*
  - (except honors track)
- http://code.google.com/p/sea-of-saf/
- Use of this repository is **required**!
- Email me (storm@cwi.nl) for access

Grading of lab assignments

- Functionality
- Tests
- Simplicity
- Modularity
- Layout and style
- Separation of concerns

Grading of la assignmen

- Functi ity
- Te
- imp icity
- Modularity
- Layout and style
- Separation f conce s

Simpli city

Some advice up-front

- Naming, layout, indentation
- Encapsulation, modularity, separation of concerns, reuse
- Don’t repeat yourself (DRY)
- Library and tool selection and use
- Unit testing

More advice

- Use asserts sensibly
- No global, static, non-final variables
- You ain't going to need it (YAGNI)
- Avoid premature optimization
- Use comments for rationale
- *Working **and** compiling code*

Grading (ctd.)

- First part: your grade is *indicative*
  - hint to improve your code
- Second part: we review all code
  - this will be your *final* grade for the lab
- Grading is on-site: *you* show your code
- Grade form will be made available

Passing this course

- Be present at all lectures
- Pass the 2 “theory” tests
- Successfully complete lab assignments
  - = beautiful, working code
- Write a good research paper
- Final grade: (*Paper* + *FinalLab*) / 2

Schedule

Date	Lecture	Tests/Grading
9-1	Introduction	
16-1	Grammars and parsing	
23-1	Domain-specific languages	Part I
30-1	Code quality	Test I
6-2	Debugging	
13-2	*DSL Digital Forensics*	Test II
20-2	*Linguistic abstraction for the Web*	Part II
27-2	Extra paper writing time*	due: 4-3-12

*I'll be in London during this week

Concluding

- All information is in Blackboard under *Course Information*
- Primary contact = me (storm@cwi.nl)

What's next

- study SAF
- get access to svn
- select parser generator + study it
- setup you project using Eclipse
- start reading papers
- start programming!