Software Architecture and Engineering

Introduction

Peter Müller

Chair of Programming Methodology

Spring Semester 2014

ETH zürich

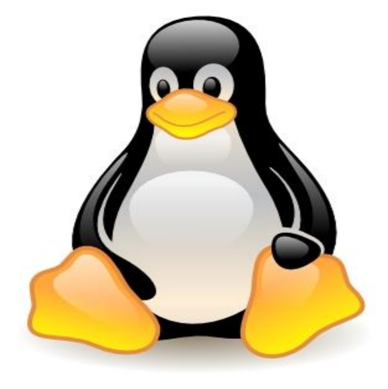
1. Introduction

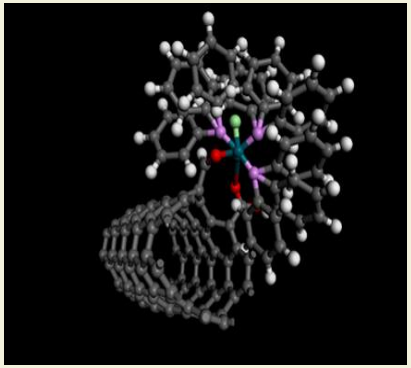
1.1 Software Failures

1.2 Challenges

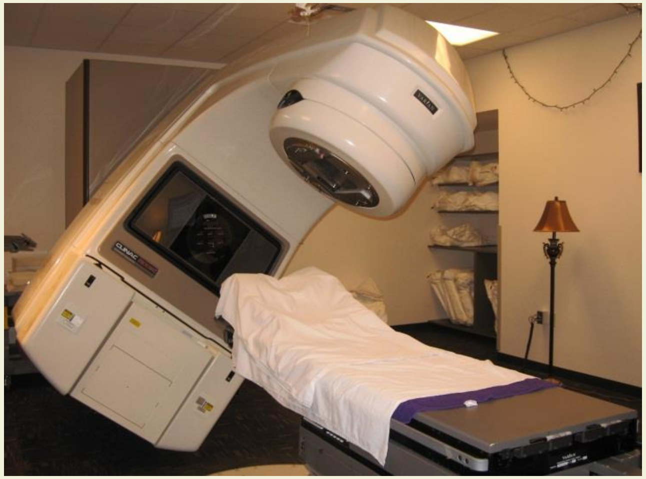
1.3 Solution Approaches (Course Outline)

Software is Everywhere





Bad Software is Everywhere



The Patriot Accident

- The Patriot missile air defense system tracks and intercepts incoming missiles
- On February 25, 1991, a Patriot system ignored an incoming Scud missile
- 28 soldiers died; 98 were injured

Patriot Bug – Rounding Error

- The tracking algorithm measures time in 1/10s
- Time is stored in a 24-bit register
  - Precise binary representation of 1/10 (non-terminating): 0.00011001100110011001100110011001…
  - Truncated value in 24-bit register: 0.00011001100110011001100
  - Rounding error: ca. 0.000000095s every 1/10s
- After 100 hours of operation error is 0.000000095s × 10 × 3600 × 100 = 0.34s
- A Scud travels at about 1.7km/s, and so travels more than 0.5km in this time

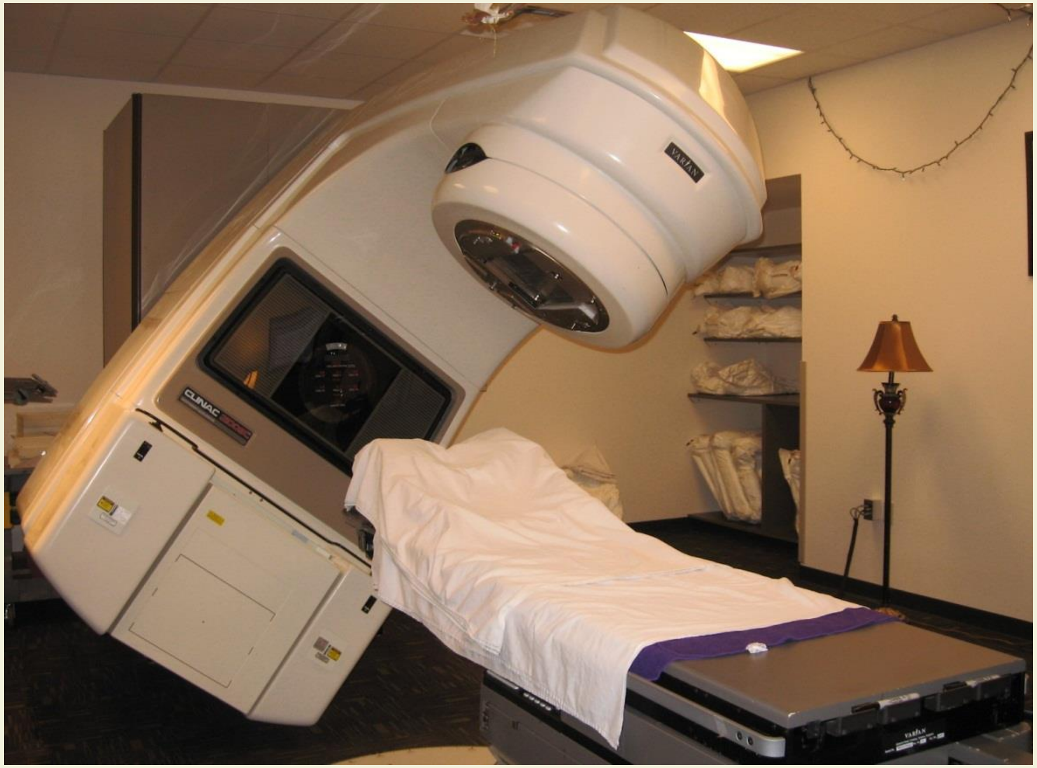
Analysis of the Patriot Accident

- Changed requirements were not considered
  - System was originally designed for much slower missiles (MACH 2 instead of MACH 5)
  - System was designed to be mobile (to avoid detection) and to operate only for a few hours at a time

- Maintenance was inadequate
  - A conversion routine with 48-bit precision was defined to cope with faster missiles, but was not called in all necessary places

The Therac-25 Accident

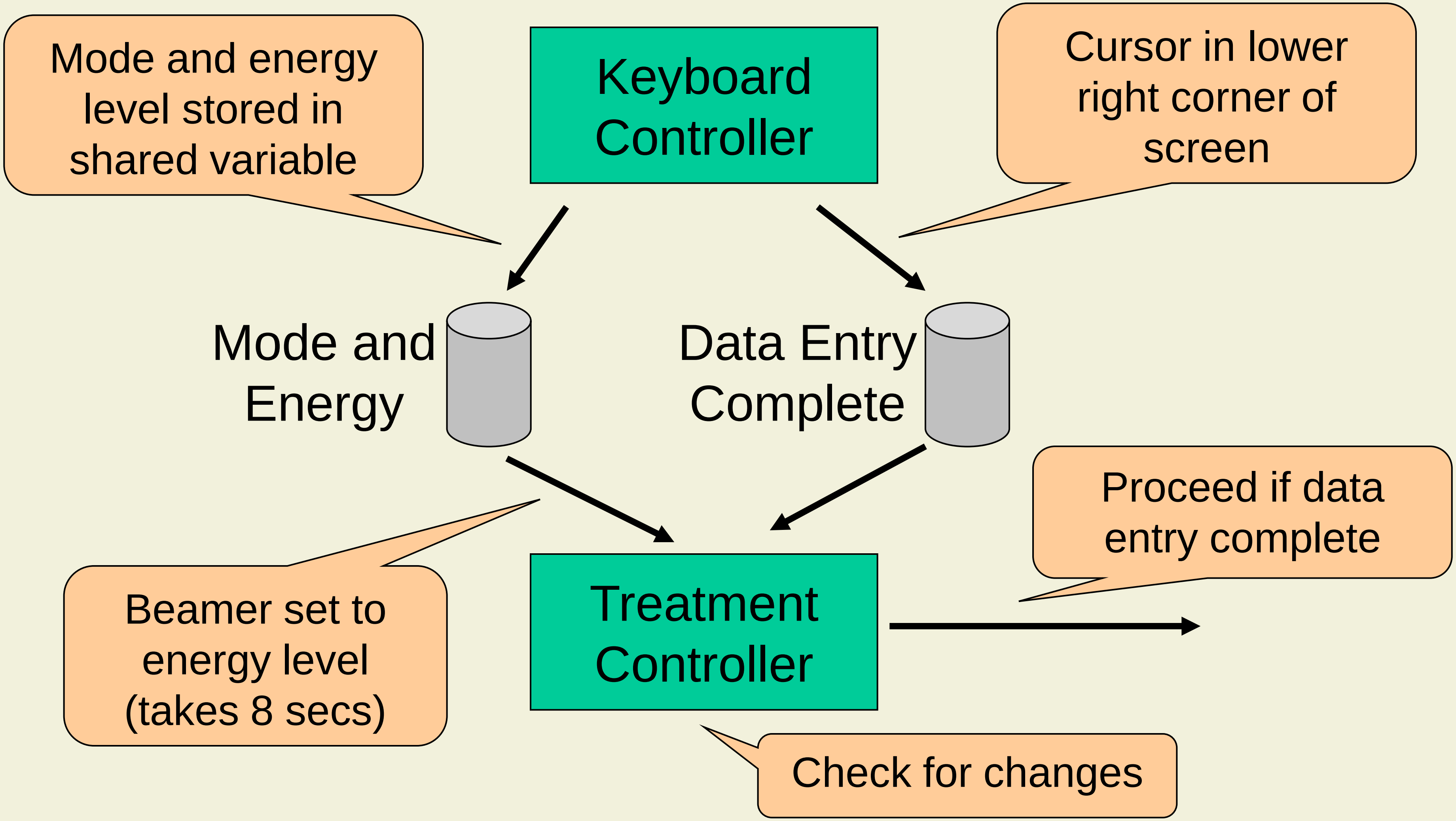
- Therac-25 is a medical linear accelerator
- High-energy X-ray and electron beams destroy tumors



Therac-25 System Design

- Therac-25 is completely computer-controlled
  - Software written in assembler code
  - Therac-25 has its own real-time operating system
- Software partly taken from ancestor machines
  - Software functionality limited
  - Hardware safety features and interlocks
- Hazard analysis
  - Extensive testing on hardware simulator
  - Program software does not degrade due to wear, fatigue, or reproduction process
  - Computer errors are caused by hardware or by alpha particles

Therac-25 Software Design



Mode and energy level stored in shared variable

Keyboard Controller

Cursor in lower right corner of screen

Mode and Energy

Data Entry Complete

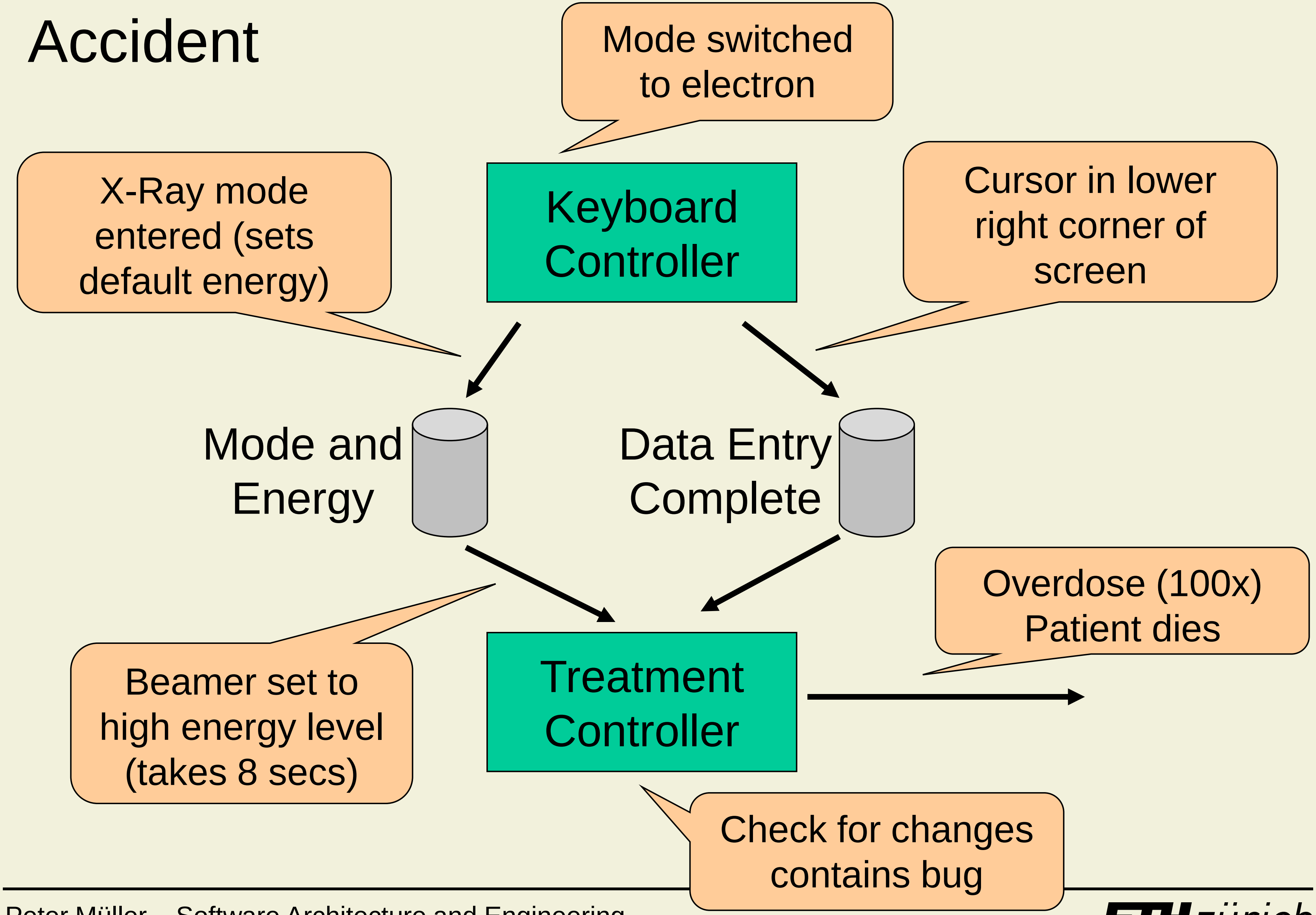
Proceed if data entry complete

Beamer set to energy level (takes 8 secs)

Treatment Controller

Check for changes

Accident



Mode switched to electron

X-Ray mode entered (sets default energy)

Keyboard Controller

Cursor in lower right corner of screen

Mode and Energy

Data Entry Complete

Overdose (100x) Patient dies

Beamer set to high energy level (takes 8 secs)

Treatment Controller

Check for changes contains bug

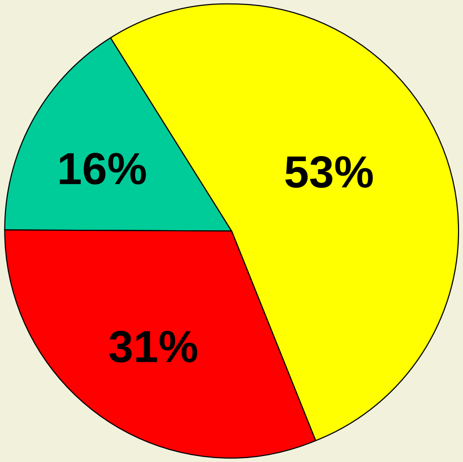
Analysis of the Therac-25 Accident

- Changed requirements were not considered
    - In Therac-25 software is safety-critical
- Design is too complex
    - Concurrent system, shared variables (race conditions)
- Code is buggy
    - Check for changes done at wrong place
- Testing was insufficient
    - System test only, almost no separate software test
- Maintenance was poor
    - Correction of bug instead of re-design (root cause)

The Windows 98 Accident

Software – a Poor Track Record

- Software bugs cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product
- 84% of all software projects are unsuccessful
  - Late, over budget, less features than specified, cancelled
- The average unsuccessful project
  - 222% longer than planned
  - 189% over budget
  - 61% of originally specified features



1. Introduction

1.1 Software Failures

1.2 Challenges

1.3 Solution Approaches (Course Outline)

Why is Software so Difficult to Get Right?

Complexity

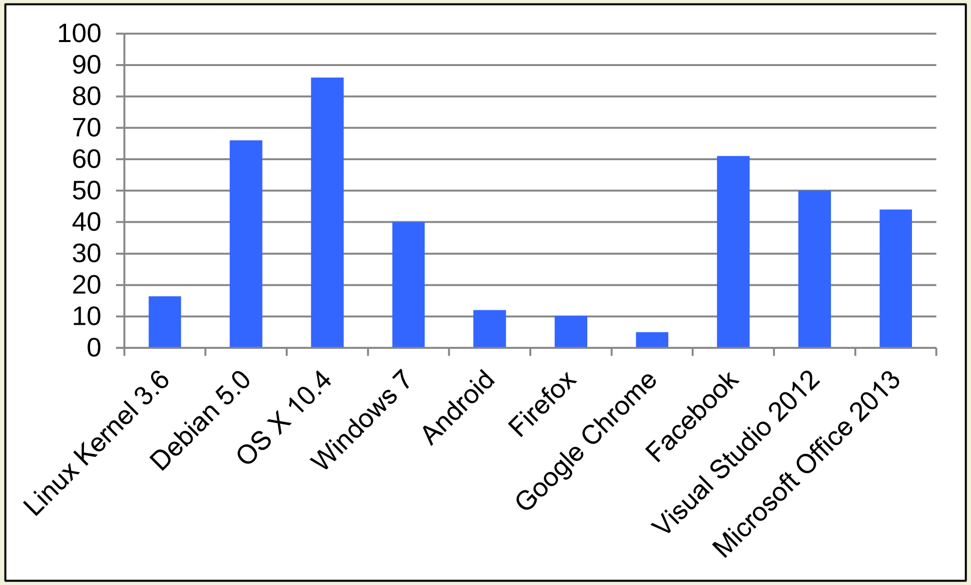
Change

Competing Objectives

Constraints

Complexity

- Modern software systems are huge
  - Created by many developers over several years

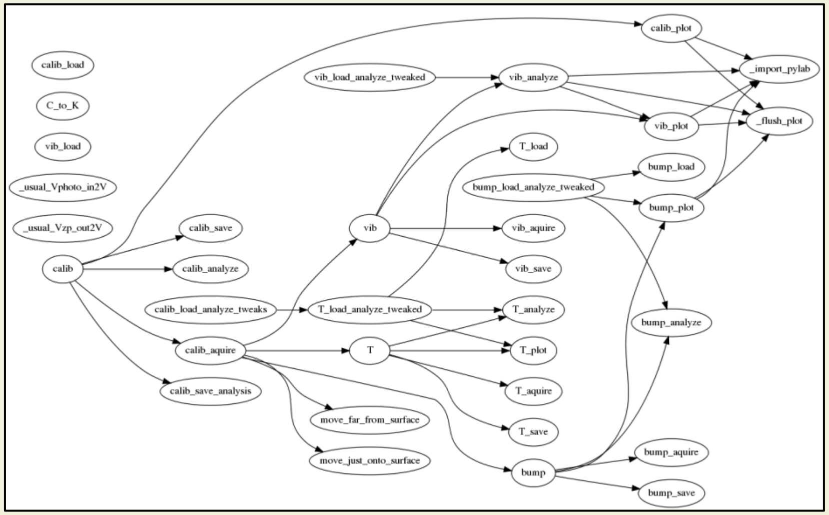


Size of software systems in MLOC

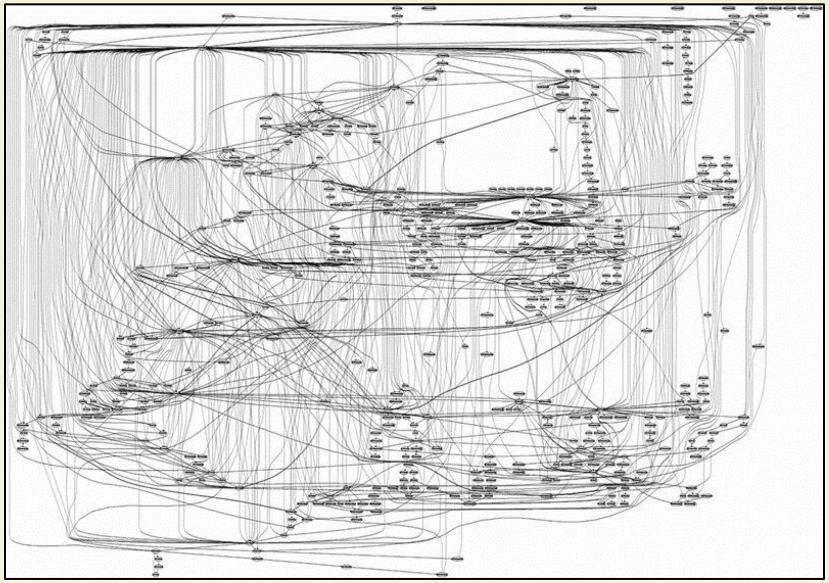
- They have a very high number of:
  - Discrete states (infinite if the memory is not bounded)
  - Execution paths (infinite if the system may not terminate)

Complexity (cont’d)

- Small programs tend to be simple



- Big programs tend to be complex
  - Complexity grows worse than linearly with size



Change

- Since software is (perceived as being) easy to change, software systems often deviate from their initial design

- Typical changes include
  - New features (requested by customers or management)
  - New interfaces (new hardware, new or changed interfaces to other software systems)
  - Bug fixing, performance tuning

- Changes often erode the structure of the system

Competing Objectives: Design Goals

Correctness

Maintainability

Performance

Verifiability

Robustness

Understandability

Scalability

Reusability

Reliability

Evolvability

Usability

Portability

Security

Repairability

Interoperability

Backward Comp.

Competing Objectives: Typical Trade-Offs

Functionality ↔ Usability

Cost ↔ Robustness

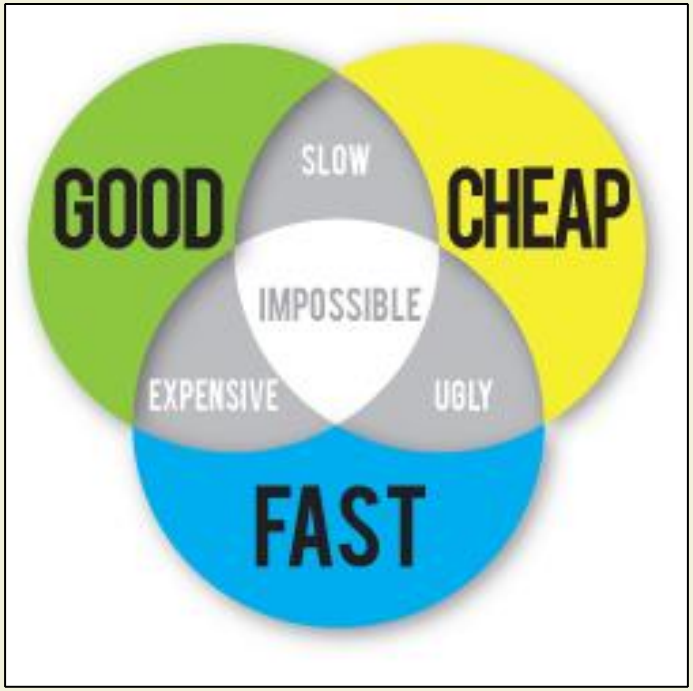
Performance ↔ Portability

Cost ↔ Reusability

Backward Compatibility ↔ Understandability

Constraints

- Software development (like all projects) is constrained by limited resources
- Budget
  - Marketing, management priorities
- Time
  - Market opportunities, external deadlines
- Staff
  - Available skills



Software Engineering

Complexity

Change

Competing Objectives

Constraints

- A collection of techniques, methodologies, and tools that help with the production of
  - a high quality software system
  - with a given budget
  - before a given deadline
  - while change occurs

[Brügge]

1. Introduction

1.1 Software Failures

1.2 Challenges

1.3 Solution Approaches (Course Outline)

Course Outline (tentative)

- We will study various principles of software engineering
- We will cover both established practices and innovative approaches
- We will emphasize software reliability

Part I: Software Design

- Modeling
- Design principles
- Architectural & design patterns

Part II: Testing

- Functional and structural testing
- Automatic test case generation
- Dynamic program analysis

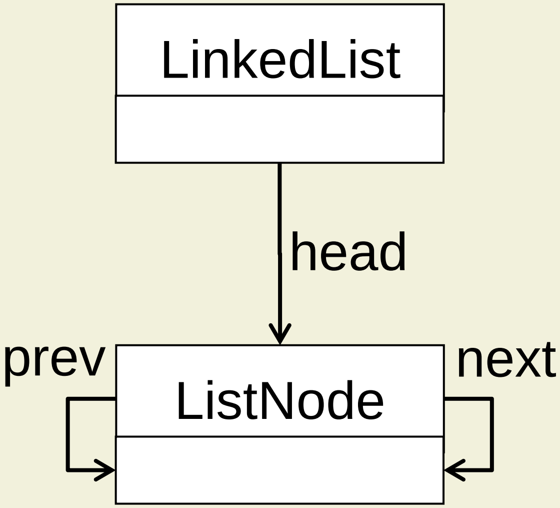
Part III: Static Analysis

- Mathematical foundations
- Abstract interpretation
- Practical applications

Overview: Modeling

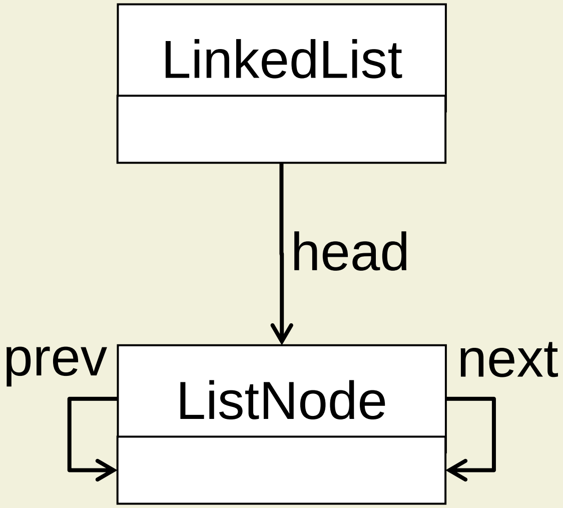
- For non-trivial systems, source code is too complex to reason
- Abstract models may simplify communication and reasoning

.NET LinkedList has 770 lines of code



Overview: Formal Modeling

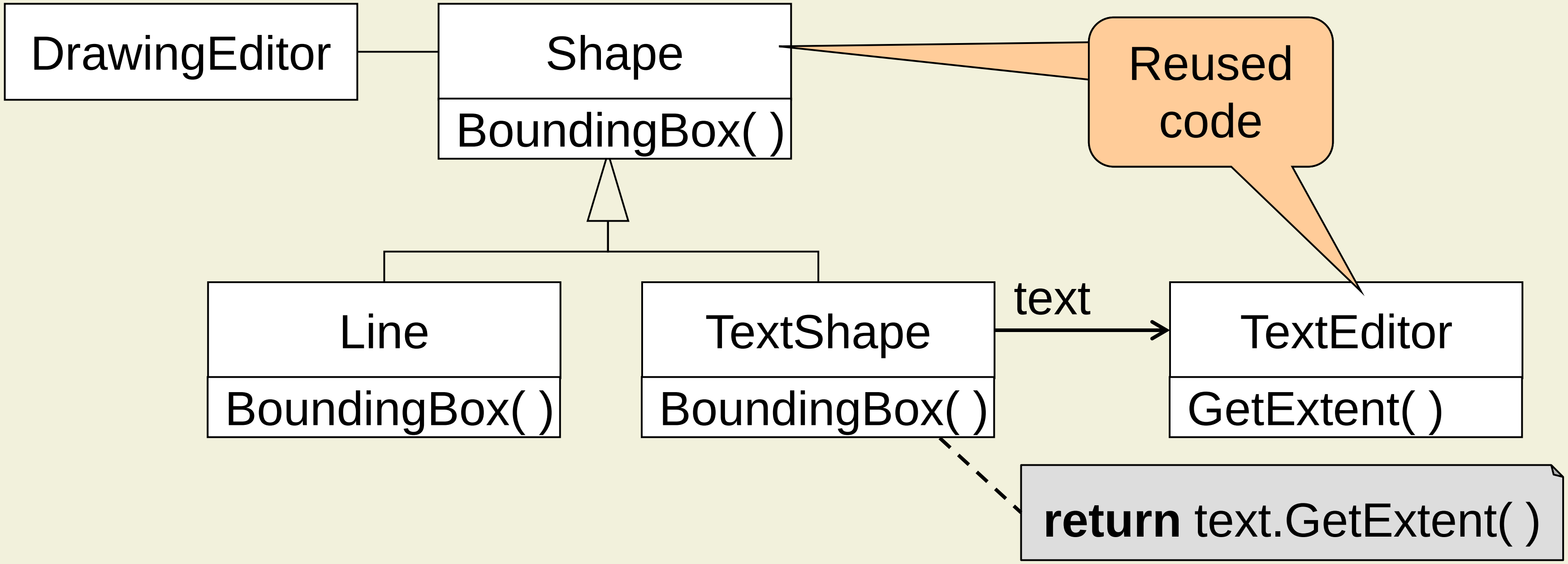
- In contrast to informal models, formal models enable precision and better tool support



```
sig LinkedList {
  head: ListNode
}

sig ListNode {
  next: ListNode,
  prev: ListNode
}

fact { all n: ListNode | n.next.prev = n }

pred show { }

run show for 5 but 2 LinkedList
```

Overview: Patterns

- Design problem:
  How to fit a reused class into a class hierarchy?



- Patterns are general, reusable solutions to commonly occurring design problems

Overview: Functional Testing

- Functional testing focuses on input/output behavior
- Given the desired functionality of a program, how to select input values to test it?

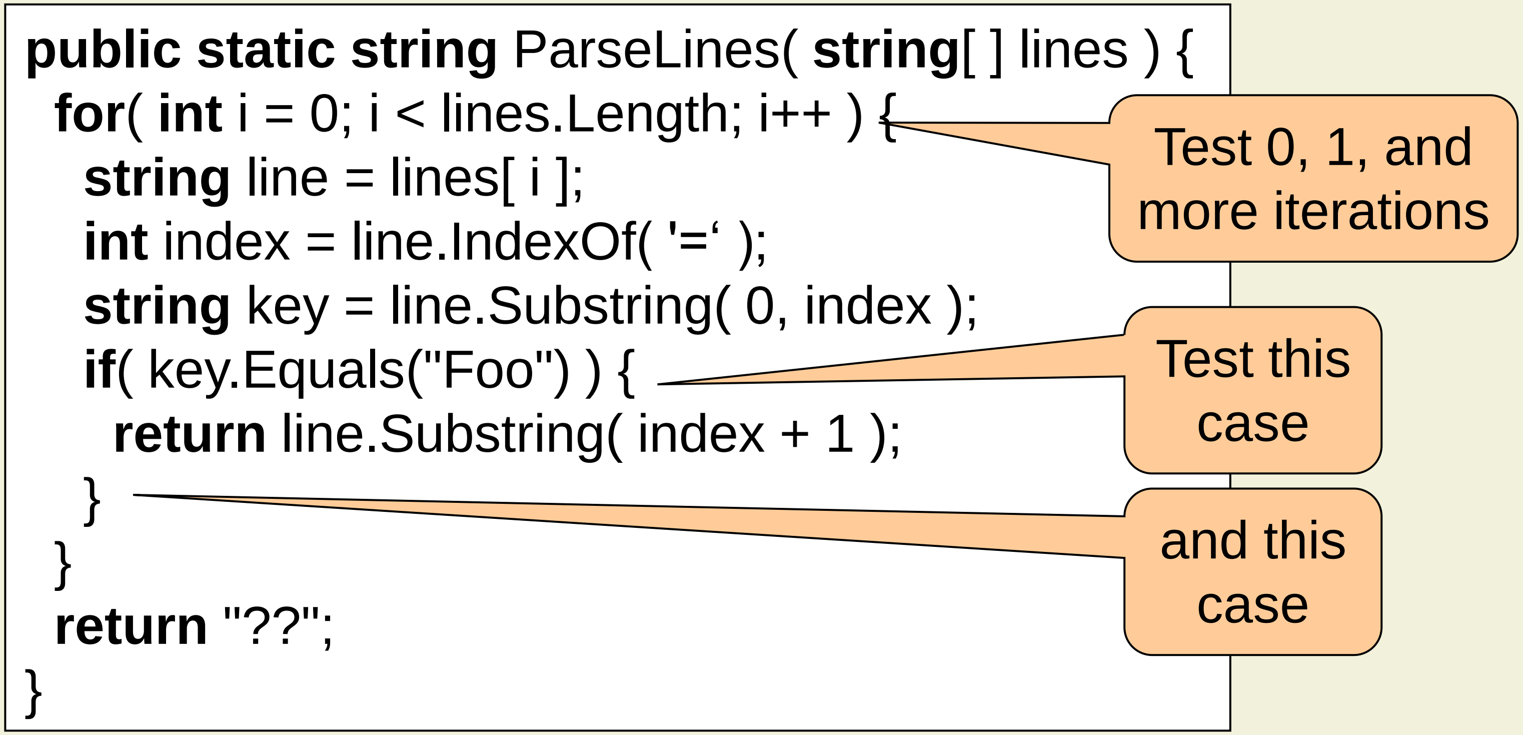
Specification:
Search for the first occurrence of
"Foo=*VALUE*" in lines and return *VALUE*.

public static string ParseLines(**string**[ ] lines)

- Try at least:
  - Arrays with one, more than one, and no matching strings
  - Corner cases: null, arrays containing null, “Foo=“

Overview: Structural Testing

- Use design knowledge about algorithms and data structures to determine test cases that exercise a large portion of the code



```
public static string ParseLines( string[ ] lines ) {
  for( int i = 0; i < lines.Length; i++ ) {
    string line = lines[ i ];
    int index = line.IndexOf( '=‘ );
    string key = line.Substring( 0, index );
    if( key.Equals("Foo") ) {
      return line.Substring( index + 1 );
    }
  }
  return "??";
}
```

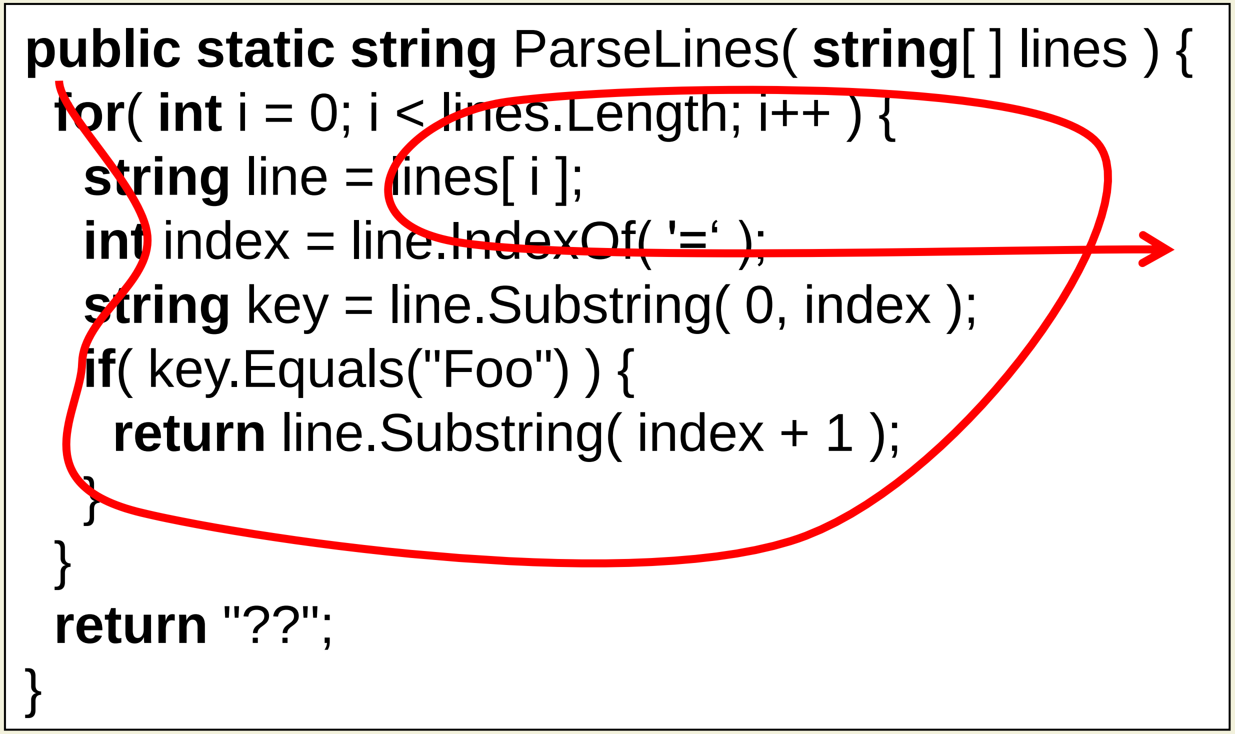
Test 0, 1, and more iterations

Test this case

and this case

Overview: Automatic Test Case Generation

- Automatically determine inputs that execute a given path through the program

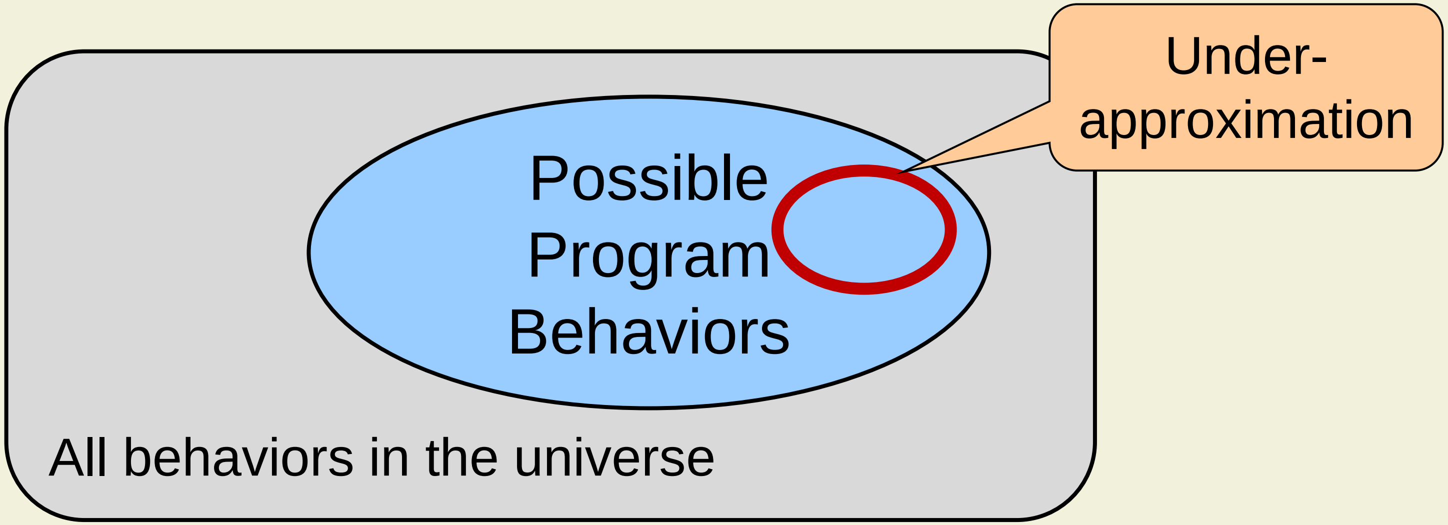


```
public static string ParseLines( string[ ] lines ) {
  for( int i = 0; i < lines.Length; i++ ) {
    string line = lines[ i ];
    int index = line.IndexOf( '=‘ );
    string key = line.Substring( 0, index );
    if( key.Equals("Foo") ) {
      return line.Substring( index + 1 );
    }
  }
  return "??";
}
```

- Suitable test input: [“Bar=XX”, **null**]

Overview: Dynamic Program Analysis

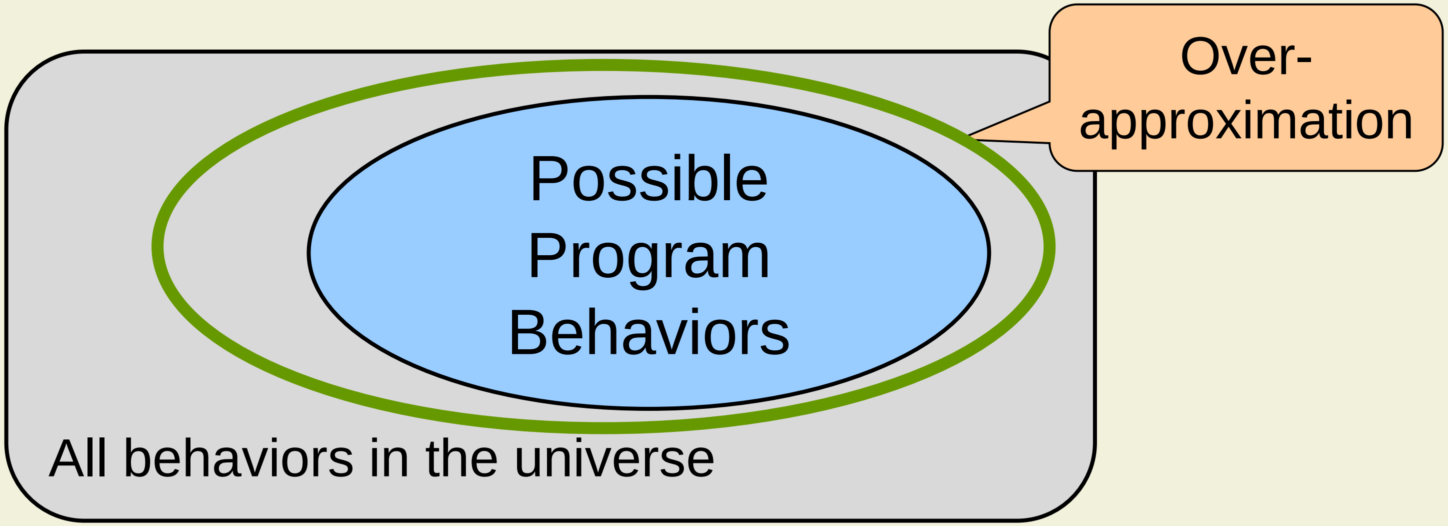
- Dynamic analyses focus on a subset of program behaviors and prove they are correct



- Testing is a special case of dynamic analysis
- More interesting cases include data race detection, memory safety, and API usage rules

Overview: Static Program Analysis

- Static analyses capture all possible program behaviors in a mathematical model and prove properties of this model



Lecturers

- First half of the course is taught by Peter Müller
  - Design, functional and structural testing
- Second half is taught by Martin Vechev
  - Automatic test case generation,
    static and dynamic analysis

Projects

- There will be two projects to help you master the techniques introduced in lectures:
  1. Build a tool that generates code and test data from design models
  2. Build a program analyzer
- Done in a group of 2 or 3, never 1
  - Select your team soon and enter it here: http://tinyurl.com/ofs3jjw
- Details will be explained later

Organization of the Course

- Prerequisites
  - Course is self-contained
  - But it combines well with other courses:
    - Formal methods and functional programming
    - Compiler Design
- Grading
  - 20% for code and test data generation project
  - 20% for analysis project
  - 60% final exam

Course Infrastructure

- Web page:
  www.pm.inf.ethz.ch/education/courses/sae
  - Slides will be available on the web page two days before the lecture (Thursday and Monday)
- Mailing list: sae2014@sympa.ethz.ch
  - We will sign you up
  - Use your ETH mail address
  - Ask general questions on the mailing list
- Submit anonymous feedback at
  http://tinyurl.com/ogbvbfx

Exercise Sessions

- Monday, 15:00-18:00
  - Andrei Dan (CHN G22)
  - Petar Tsankov (NO D11)
  - Valentin Wüstholz (ML H34.3)
- Wednesday, 15:00-18:00
  - Dimitar Dimitrov (ML F40)

Can we move this to Monday?

- Sign up at http://tinyurl.com/o8ctrog
- Exercises start next week (Feb. 24 or 26)!

Don’t Forget!

- Sign up for the exercises at
  http://tinyurl.com/o8ctrog

- Select your project team and enter it at
  http://tinyurl.com/ofs3jjw