Rigorous Software Engineering

Introduction

Prof. Zhendong Su

(based on slides from Prof. Peter Müller)

Spring 2019

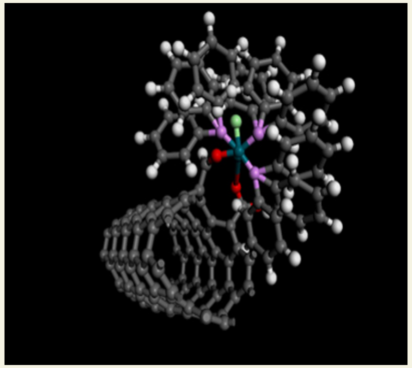
ETH zürich

苏振东 (Su Zhen Dong)

- How to pronounce “Zhendong”?
  - Try “Jendong” (close enough)
- Places lived
  - Hebei, Shanghai, Wisconsin, Texas, California, Zurich
- Places studied/worked
  - Fudan, UT Austin, UC Berkeley, UC Davis, ETH Zurich
- Research interests (AST Lab AST ADVANCED SOFTWARE TECHNOLOGIES @ ETH)
  - Methodologies & techniques for reliable/secure software
  - EdTech for K-12 and CS education
  - AI reliability, security, performance, usability

1. Introduction

Software is Everywhere (and Eating the World)



Bad Software is Everywhere

The Patriot Accident

- The Patriot missile defense system tracks & intercepts incoming missiles

- On Feb. 25, 1991, a Patriot system ignored an incoming Scud missile

- Aftermath
  - 28 soldiers died
  - 98 injured

Patriot Bug – Rounding Error

- The tracking algorithm measures time in 1/10s (tick)
- Time is stored in a 24-bit fixed-point register
  - Precise binary representation of 1/10 (infinite):
    0.0001100110011001100110011001100…
  - Truncated value in 24-bit fixed-point register:
    0.00011001100110011001100
  - Rounding error: ~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

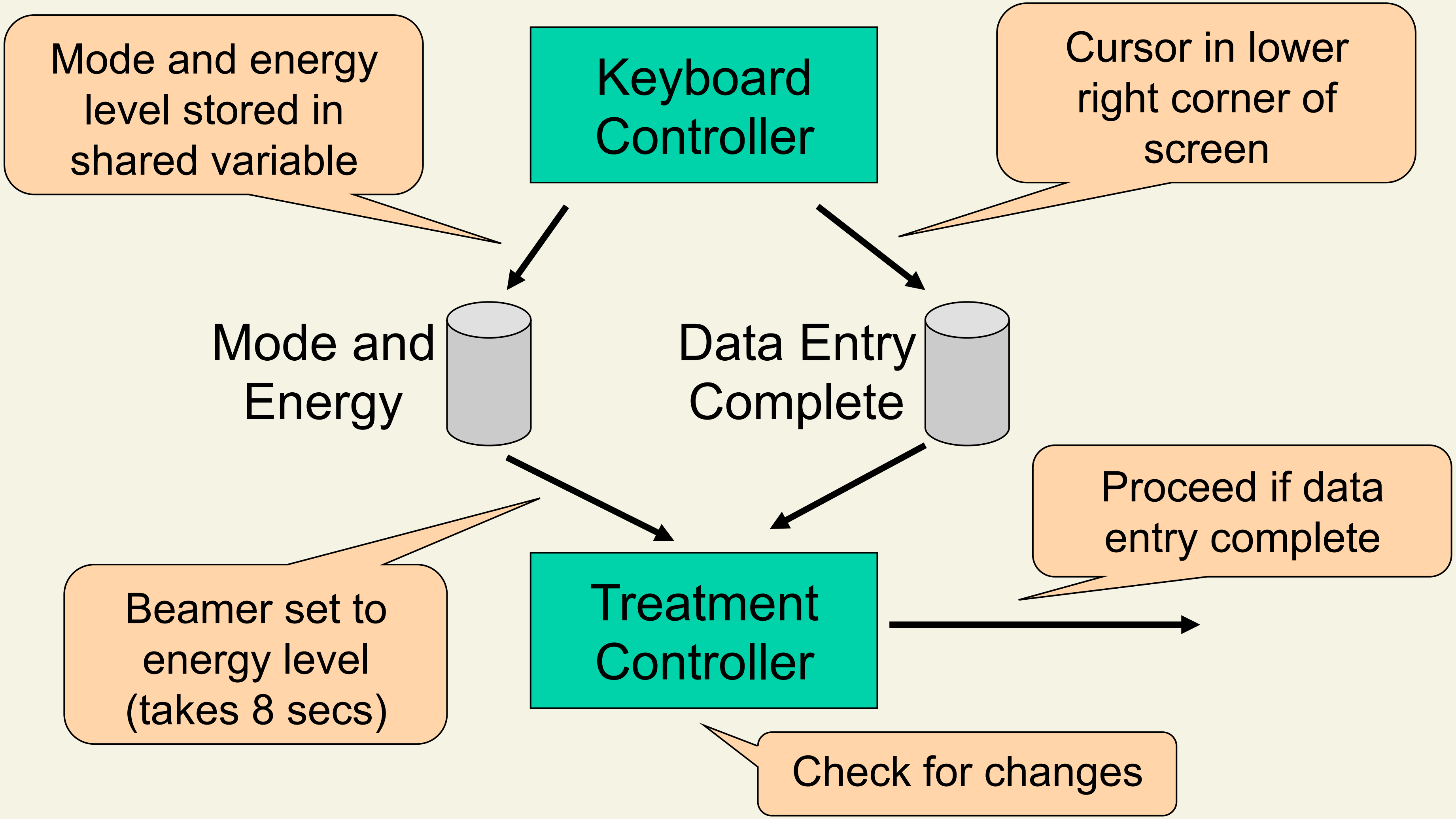
The Therac-25 Accident

- Therac-25 is a medical linear accelerator
- High-energy X-ray & electron beams destroy tumors
- 6 people died or were seriously injured during 1985-1987

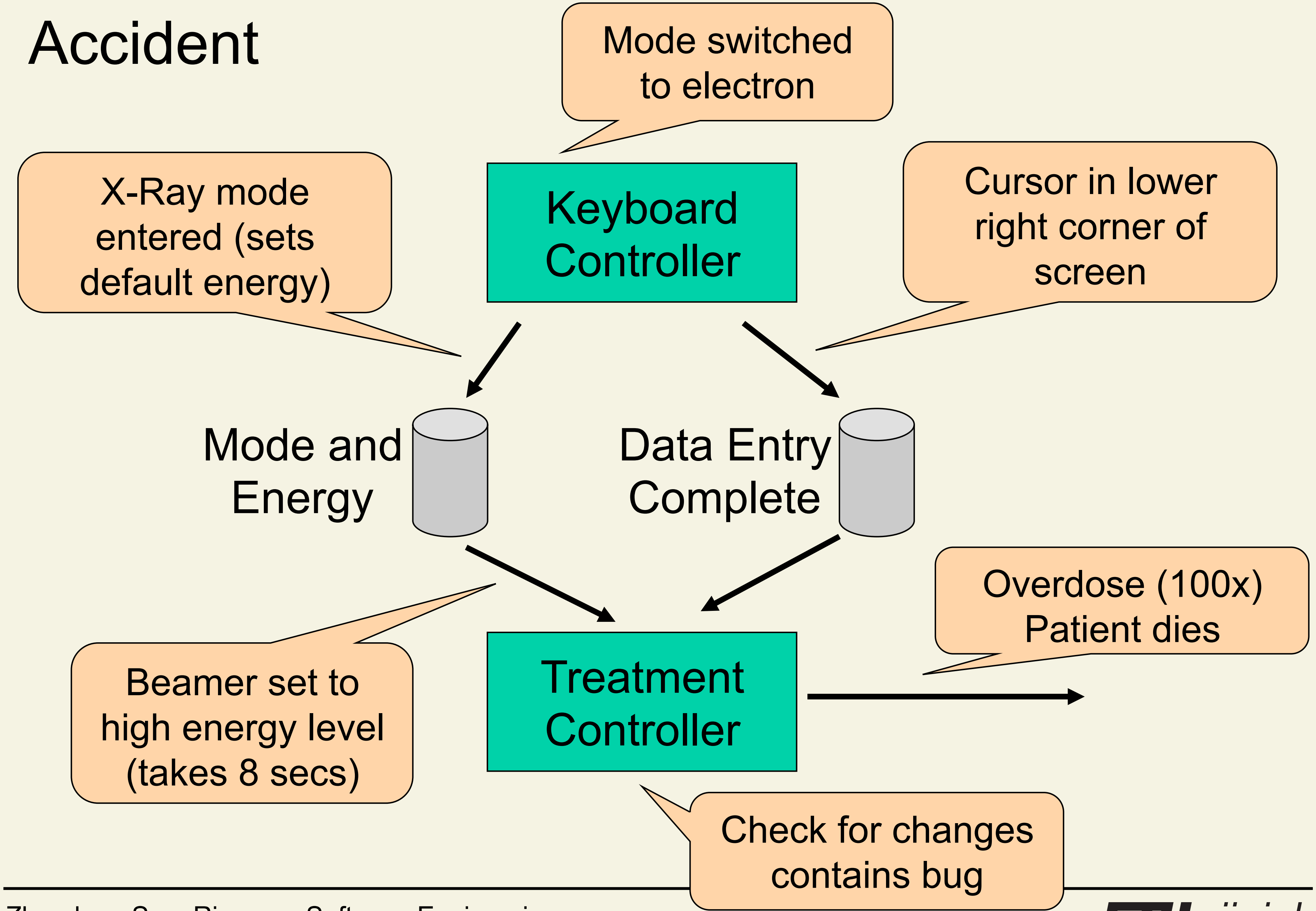
Therac-25 System Design

- Therac-25 is completely computer-controlled
  - Software written in assembly 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 due to hardware or alpha particles

Therac-25 Software Design



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

14 Years Later

Google Translate Mistranslation

https://www.zoo.ch/de/zoobesuch/tickets-preise

TICKETS & PRICES

INDIVIDUAL ENTRY

Single entries from our online ticket shop are **only valid for the selected date** .

category	Price / person
Adults (from 21 years)	CHF 26.- BUY ONLINE
Teenagers (16-20 years)	CHF 21.- BUY ONLINE
Children (6-15 years)	CHF 13.- BUY ONLINE
Children under 6 years	free
IV-recipient with valid ID	CHF 13.-
Family day card (life partner with own children, 6-15 years)	CHF 71.- BUY ONLINE

Children up to the age of 15 are given free admission to the zoo on presentation of a valid ID.

SPECIAL OFFERS ON MONDAY

Except holidays.

category	Price / person
Students (21-29 years)	CHF 21.-
People over 64 years	CHF 21.-

30% DISCOUNT WITH RAILAWAY

From February 11 to March 17, 2019, SBB RailAway will benefit from up to 30% off travel on public transport and entry to the zoo.

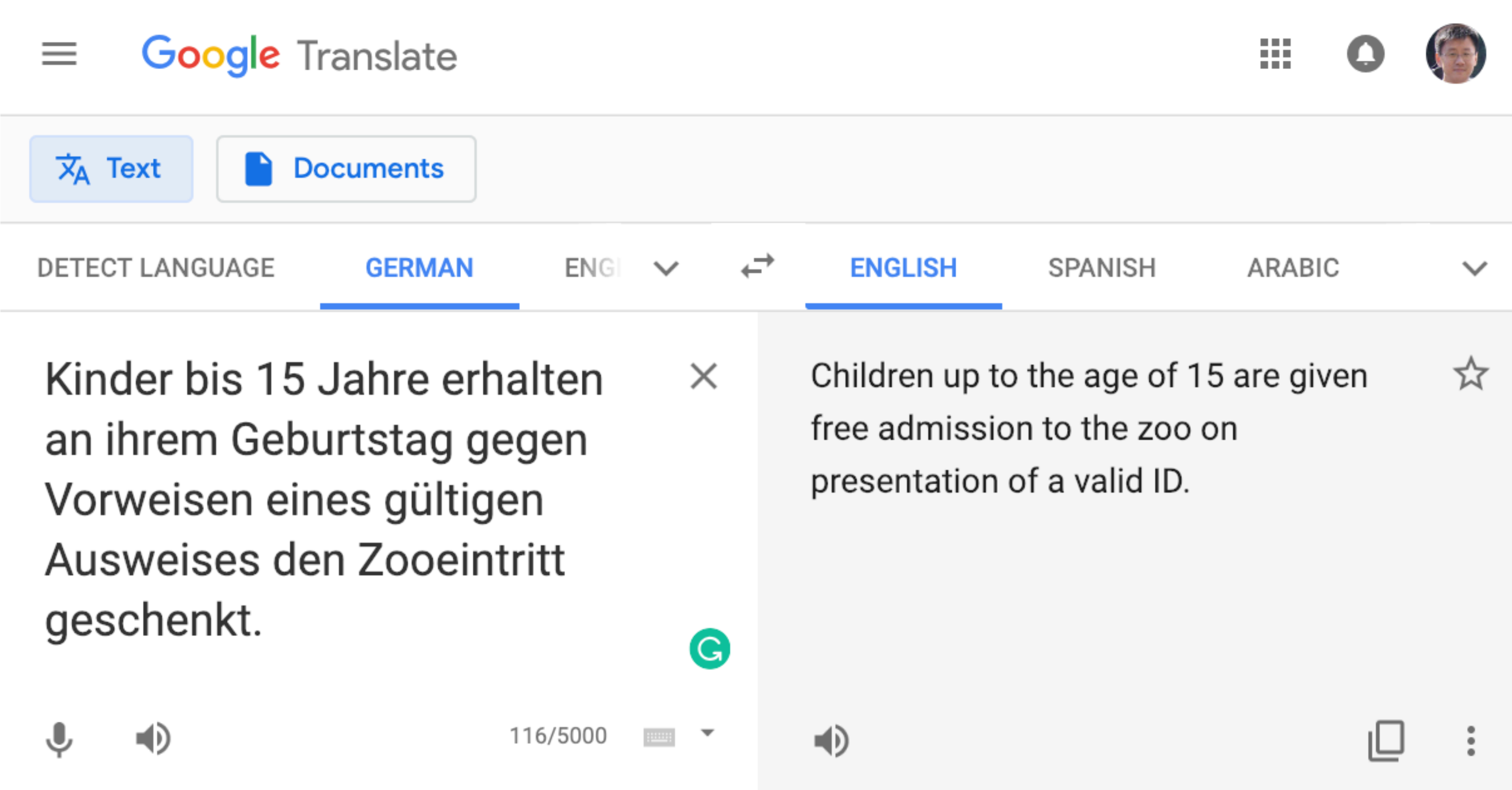
BUY NOW

ORDER VOUCHERS

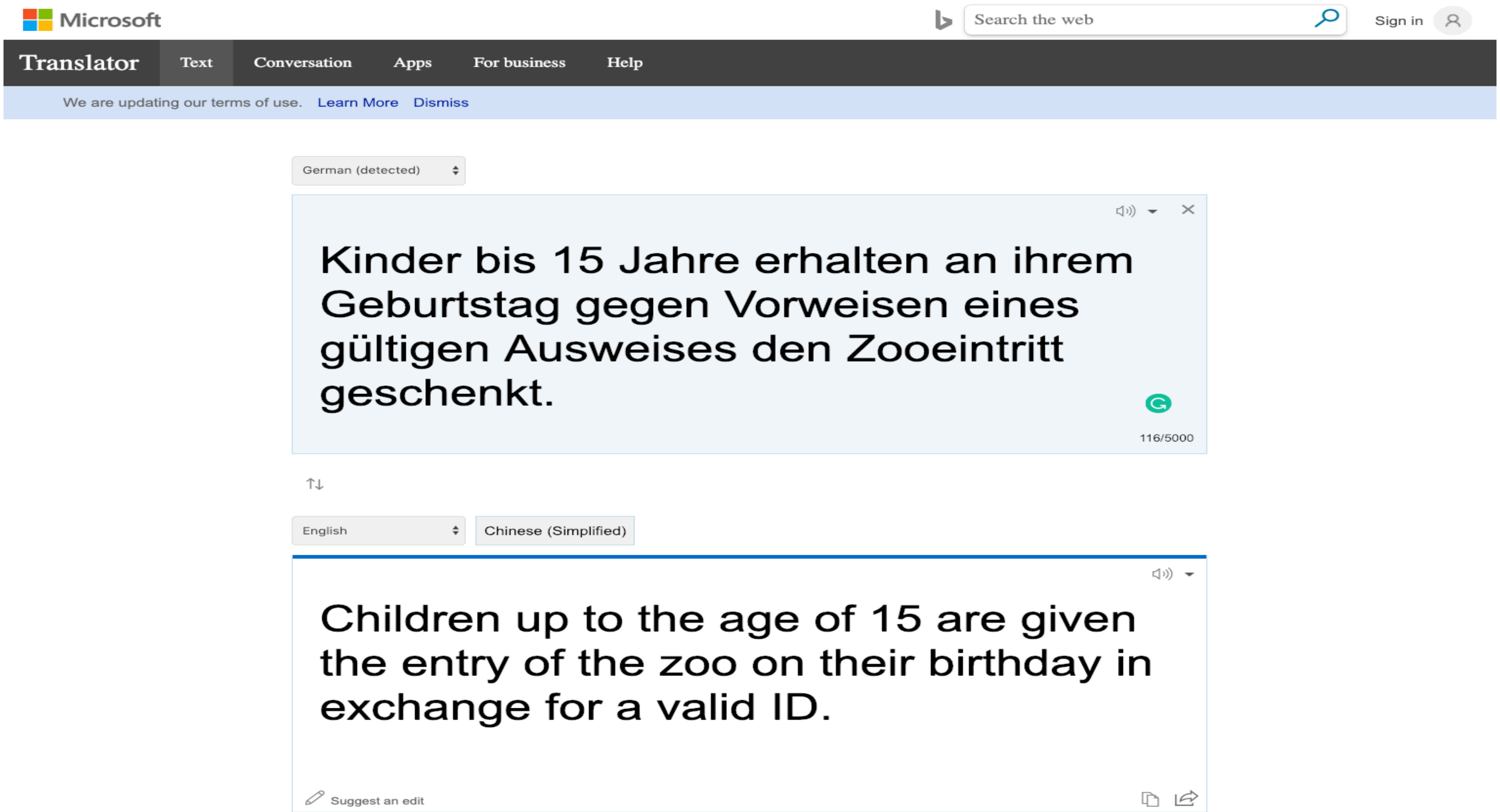
You would like to give away a single entry or an annual pass for the Zoo Zurich? Order a gift voucher in our online shop.

ORDER NOW

Google Translate Mistranslation



Google Translate Mistranslation

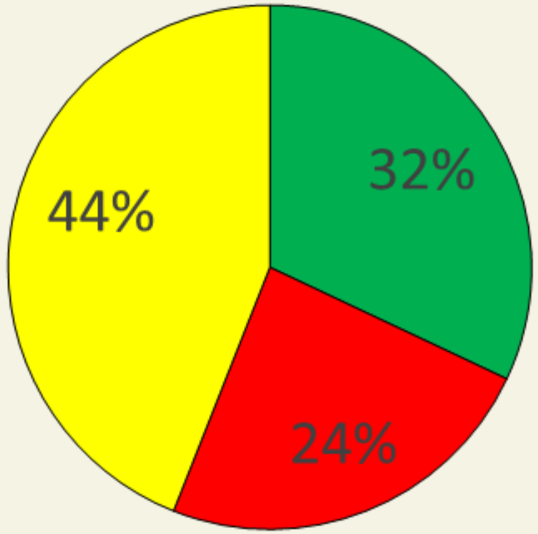


Compiler Bug

```
$ wc small.c
 2 29 92 small.c
$ clang -O1 small.c; a.out; echo $?
0
$ clang -O0 small.c; a.out; echo $?
1
$ cat small.c
int f (int p, int q) { return p > q || (p && q) ? p : q; }
int main () { return f (0, 1); }
$
```

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 [NIST, 2002]
- **68%** of all software projects are unsuccessful [Standish, 2008]
  - Late, over budget, less features than specified (**44%**); cancelled (**24%**)
- The average unsuccessful project
  - 179% longer than planned
  - 154% over budget
  - 67% of originally specified features



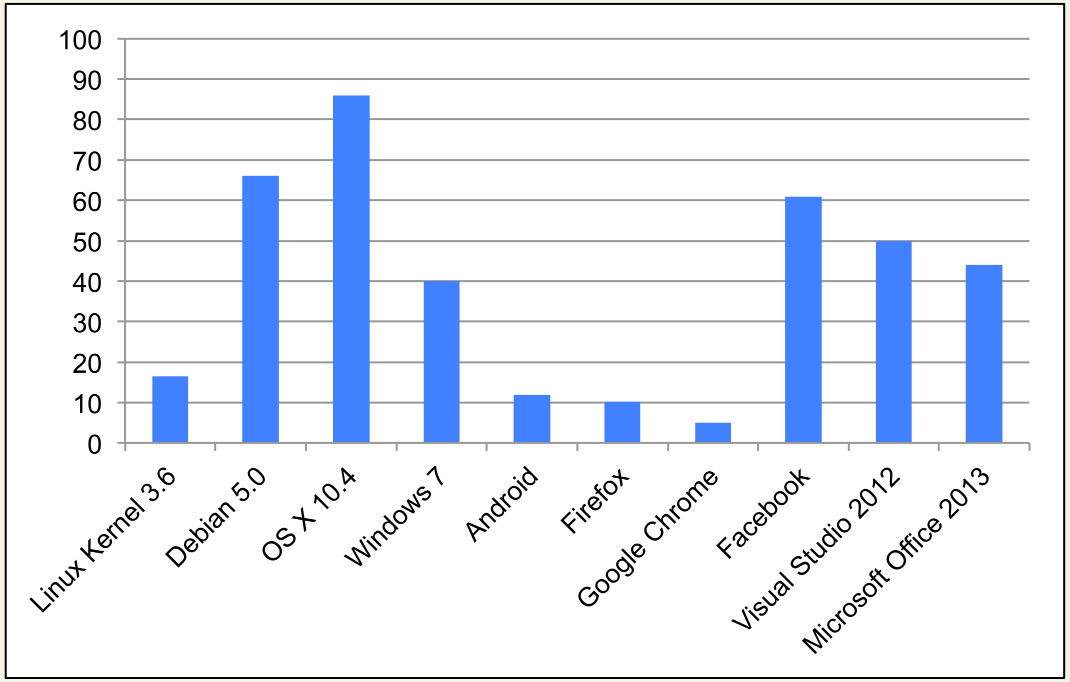
1. Introduction

Why is Software so Difficult to Get Right?

Complexity	Change
Competing Objectives	Constraints

Complexity

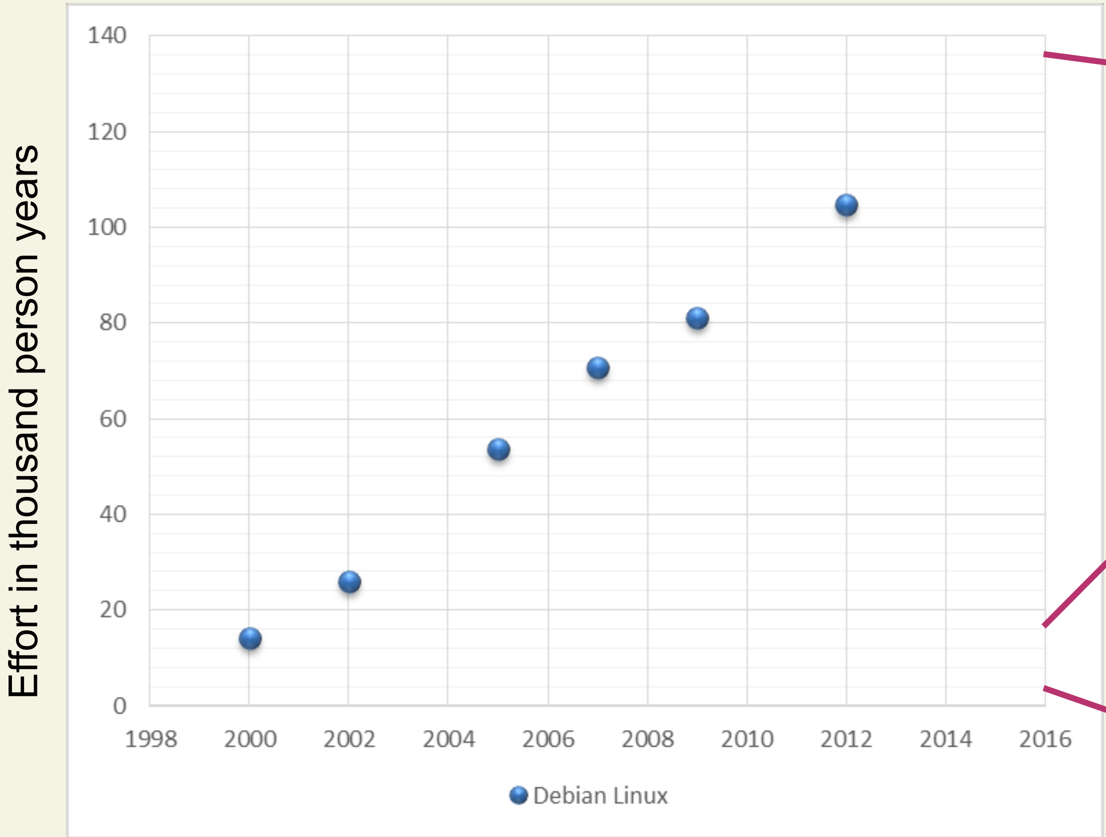
- Modern software is huge --- created by many developers over several years



Size of software systems in MLOC

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

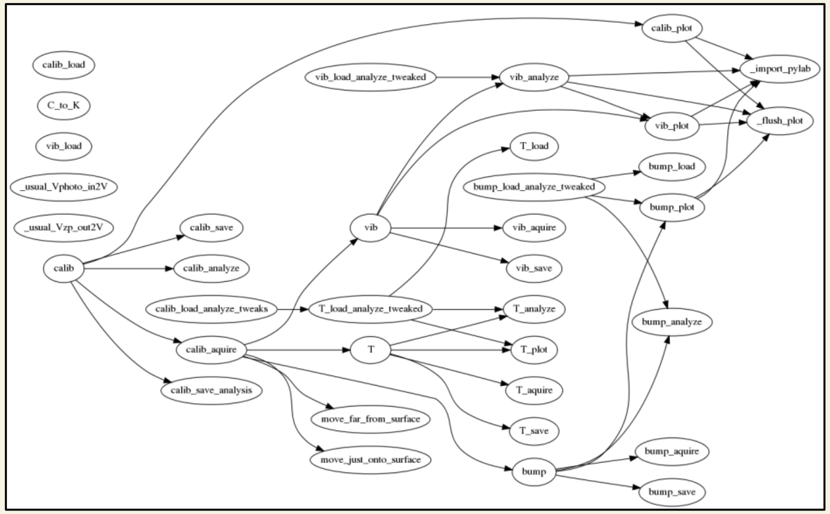
Complexity (cont'd)

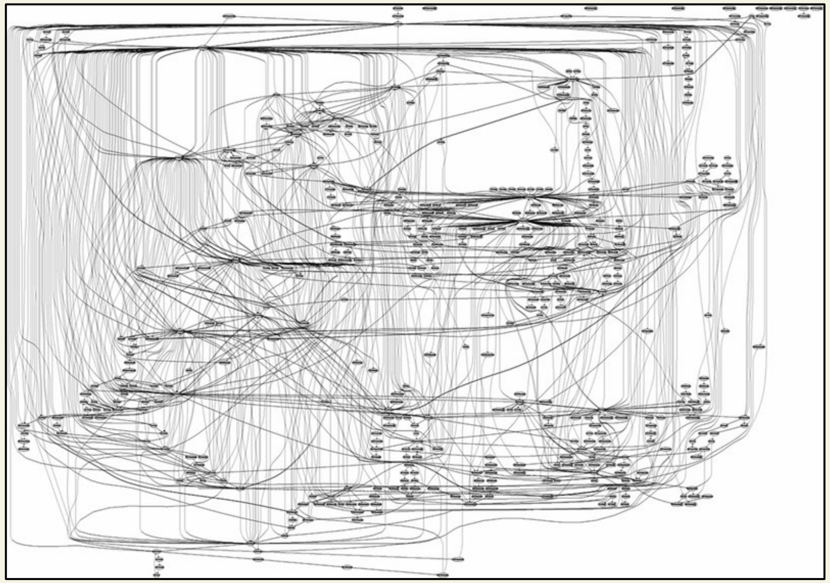


Complexity (cont'd)

- Small programs tend to be simple

- Big ones 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 & tools that help produce
  - high-quality software
  - within a given budget
  - before a given deadline
  - while change occurs

[Brügge]

1. Introduction

Course Outline (tentative)

- Study SE principles
- Cover established practices & recent innovations
- 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

Lecturers

- First half of the course is taught by Zhendong Su
  - Design & modeling
  - Functional & structural testing
- Second half is taught by Martin Vechev
  - Automated test generation
  - Static & dynamic analysis

Projects

- There will be two projects to help you learn the techniques introduced in the lectures
- Done in groups, never solo
  - Select your team soon (watch for announcement)
- 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
    - Software Engineering Seminar

- Grading
  - 30% project
  - 70% final exam

Course Infrastructure

- Web page
  https://people.inf.ethz.ch/suz/teaching/252-0216.html
  - Slides will be available on the webpage before the lecture
  - Check regularly for announcements

- Mailing list
  rse-students@lists.inf.ethz.ch (tentative)
  - We will sign you up
  - Ask general questions on the mailing list

Exercise Sessions

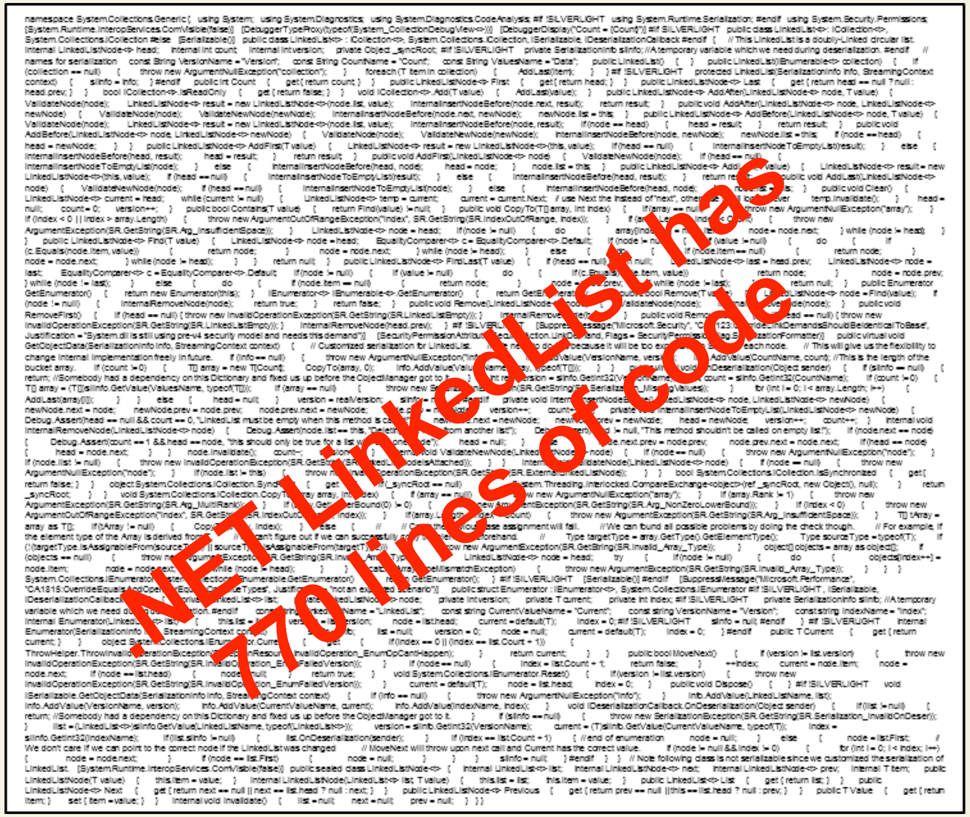
- Monday, 13:00-16:00, CHN D 44
- Tuesday, 15:00-18:00, CHN D 48
- Tuesday, 15:00-18:00, HG D 3.1
- Tuesday, 15:00-18:00, ML E 12
- Thursday, 15:00-18:00, ETZ F 91

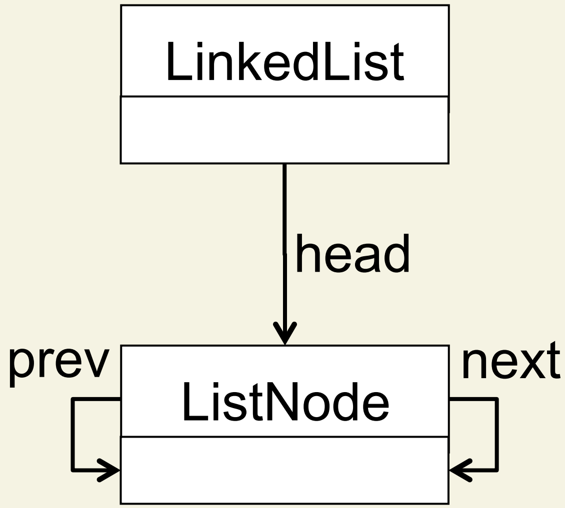
- We will sign you up, based on your input

- **Exercises start next week**

Overview: Modeling

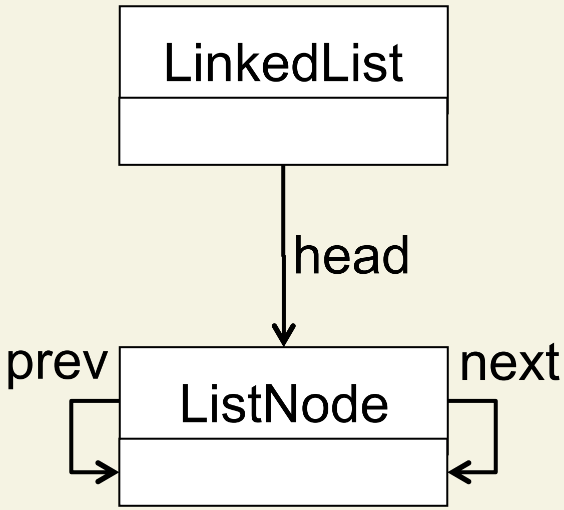
- Code of nontrivial systems is too complex to reason about
- Abstract models may simplify communication and reasoning





Overview: Formal Modeling

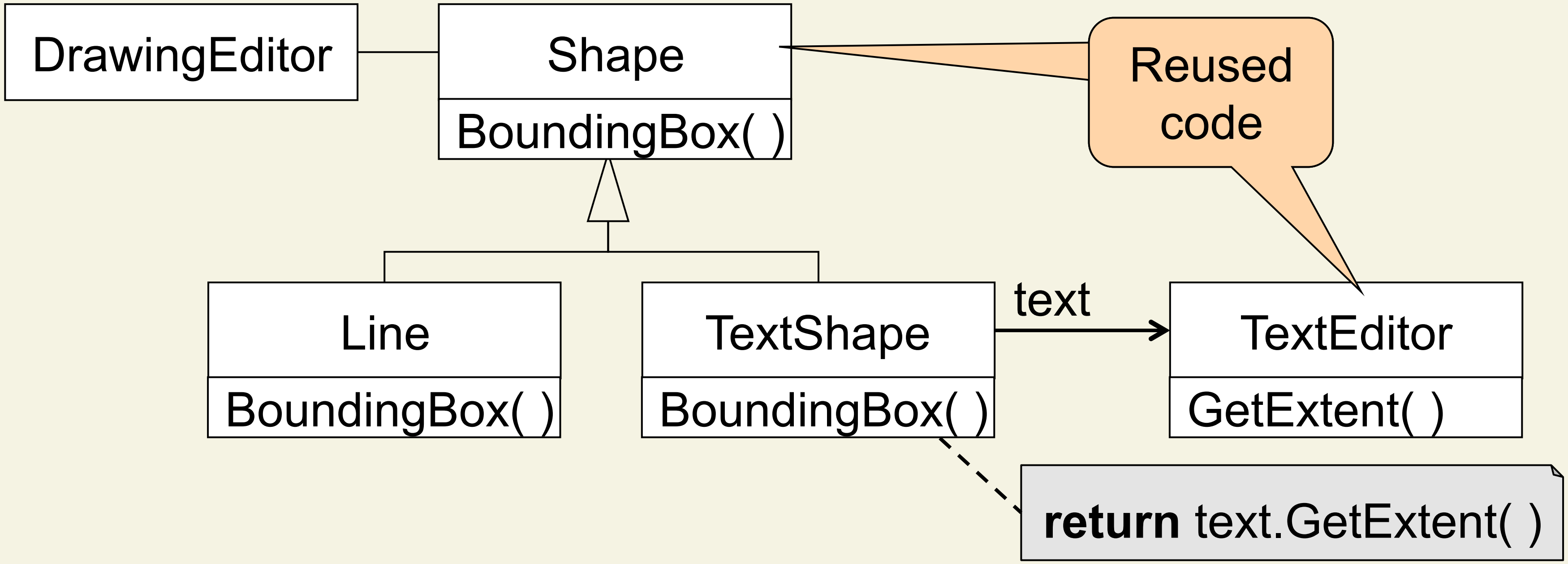
- In contrast to informal models, formal models enable precision and better tool support




```
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

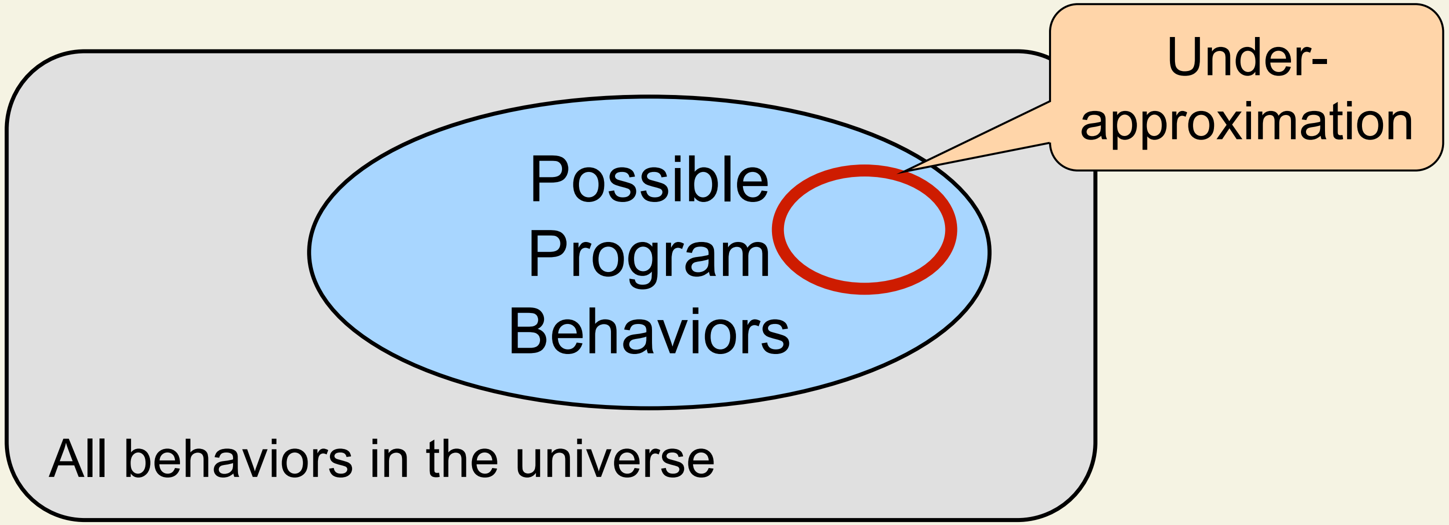
- Automatically determine inputs that execute a given path through the program

```
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
- Other applications 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