Software System Design and Implementation

Admin & Motivation & Some History

Gabriele Keller

Admin: Liam O'Connor-Davies

The University of New South Wales
School of Computer Science and Engineering
Sydney, Australia

COMP3141 17s1

Admin

- Course website is the main source of information for this course:
    - www.cse.unsw.edu.au/~cs3141
- Piazza as course forum
    - you should receive invitation shortly
- Consultation time
    - Monday after the lecture - please send an email
    - otherwise, by appointment

Lectures

- Monday lectures will start at **9:15** and we skip the break
- Lectures will be recorded via Echo
- We're using 'OneNote' to share class whiteboard content
  - link and invitation coming soon

Assessment

- No tutes, but regular exercises and quizzes
  - to practice and deepen understanding of lecture content
- 2 Assignments
- Final exam
  - harmonic mean between class mark and exam
  - need to pass exam (40%) to pass the course

Motivation

What this course is all about?

Software affects our lives everywhere

- Mobile phones
- Social networks
- Cars & public transport
- Multimedia devices
- Medical systems
- Crypto currency
- Obviously, on computers at home and at work

Photo by Prosper973 - http://flic.kr/p/6pECyk

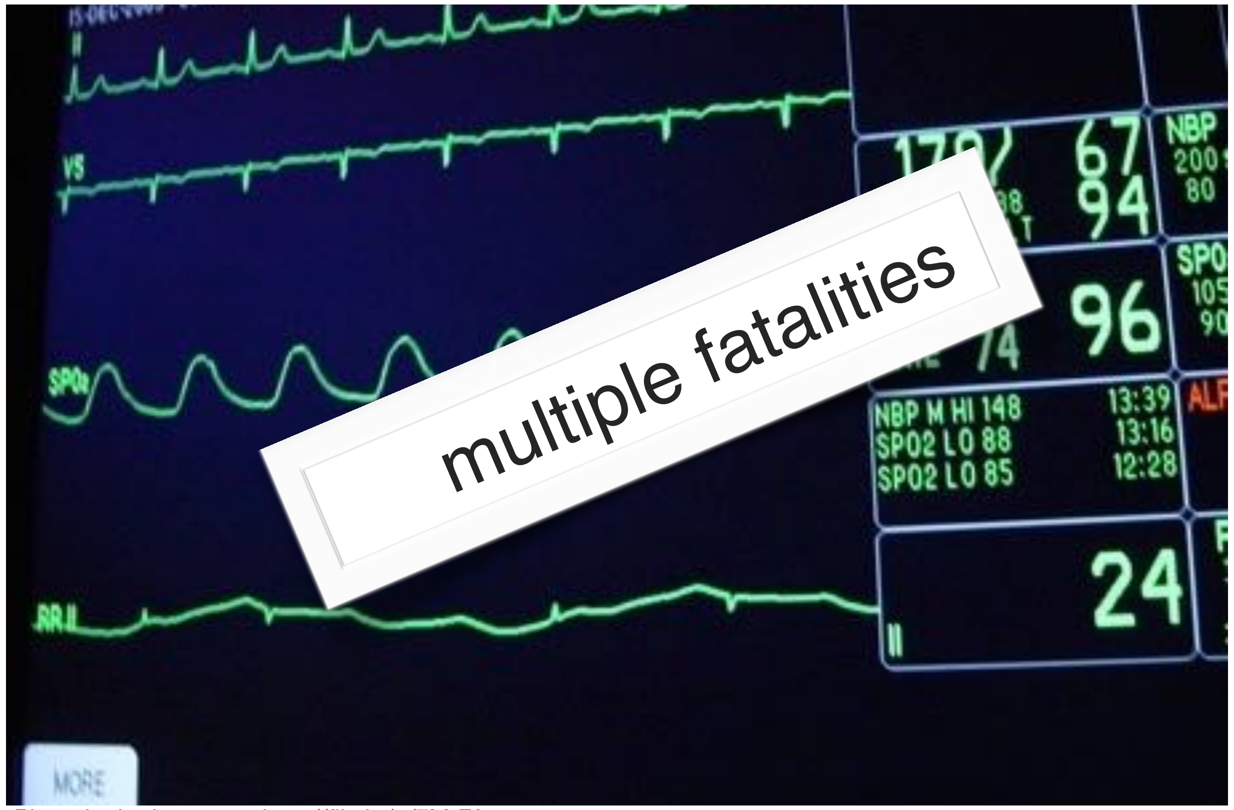
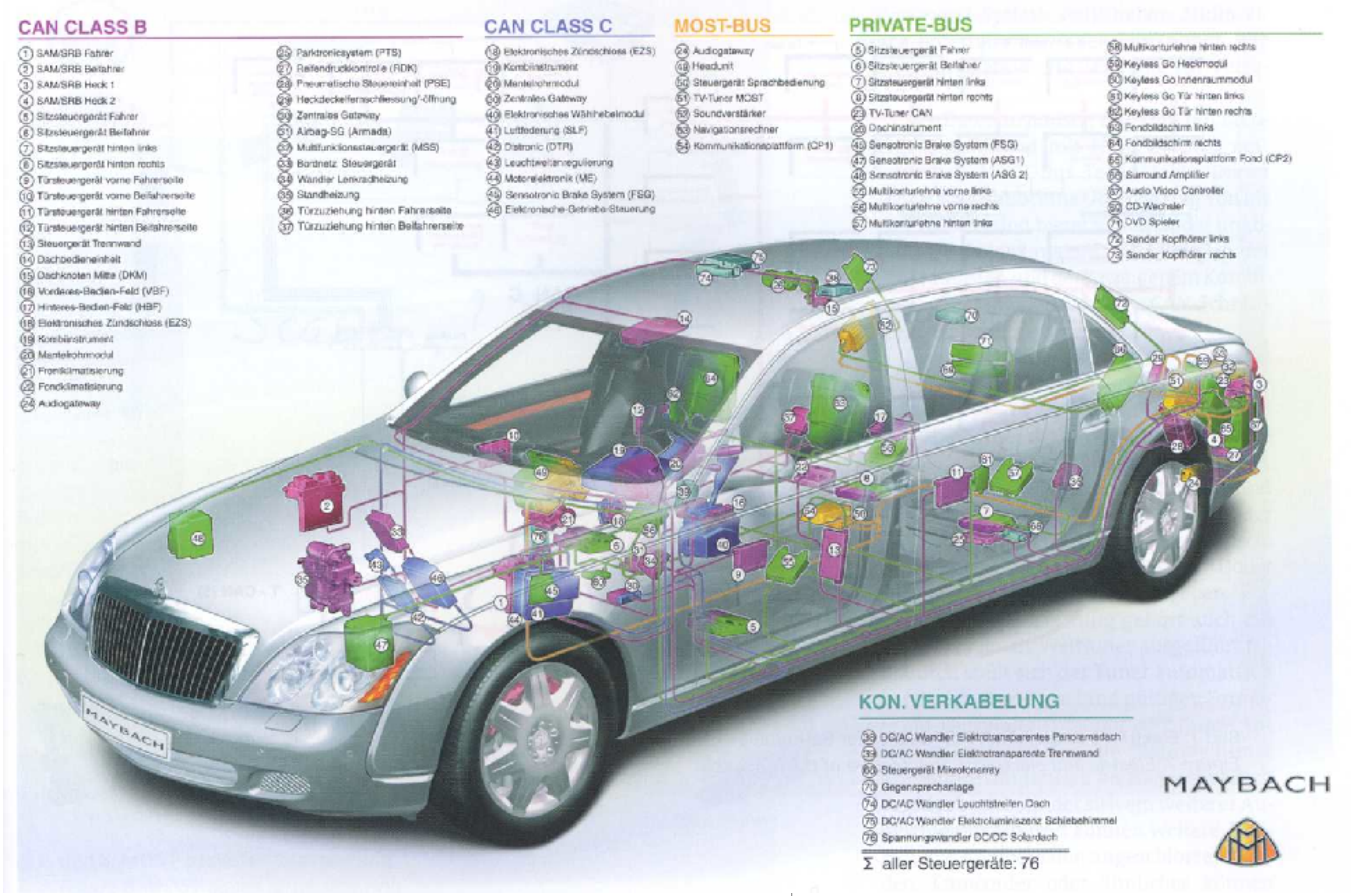


Photo by brykmantra - http://flic.kr/p/7MrFA

Photo by banger1977 - http://flic.kr/p/4nTFPQ

Software is often mission critical.

UNSW THE UNIVERSITY OF NEW SOUTH WALES SYDNEY • AUSTRALIA


CAN CLASS B
1 SAM/SRB Fahrer
2 SAM/SRB Beifahrer
3 SAM/SRB Heck 1
4 SAM/SRB Heck 2
5 Sitzsteuergerät Fahrer
6 Sitzsteuergerät Beifahrer
7 Sitzsteuergerät hinten links
8 Sitzsteuergerät hinten rechts
9 Türsteuergerät vorne Fahrerseite
10 Türsteuergerät vorne Beifahrerseite
11 Türsteuergerät hinten Fahrerseite
12 Türsteuergerät hinten Beifahrerseite
13 Steuergerät Trennwand
14 Dachbedieneinheit
15 Dachknoten Mitte (DKM)
16 Vorderes-Bedien-Feld (VBF)
17 Hinteres-Bedien-Feld (HBF)
18 Elektronisches Zündschloss (EZS)
19 Kombiinstrument
20 Mantelrohrmodul
21 Frontklimatisierung
22 Fondklimatisierung
24 Audiogateway
26 Parktronicsystem (PTS)
27 Reifendruckkontrolle (RDK)
28 Pneumatische Steuereinheit (PSE)
29 Heckdeckelfernschliessung/-öffnung
30 Zentrales Gateway
31 Airbag-SG (Armada)
32 Multifunktionssteuergerät (MSS)
33 Bordnetz Steuergerät
34 Wandler Lenkradheizung
35 Standheizung
36 Türzuziehung hinten Fahrerseite
37 Türzuziehung hinten Beifahrerseite
CAN CLASS C
18 Elektronisches Zündschloss (EZS)
19 Kombiinstrument
20 Mantelrohrmodul
30 Zentrales Gateway
40 Elektronisches Wählhebelmodul
41 Luftfederung (SLF)
42 Distronic (DTR)
43 Leuchtweitenregulierung
44 Motorelektronik (ME)
45 Sensotronic Brake System (FSG)
46 Elektronische-Getriebe-Steuerung
MOST-BUS
24 Audiogateway
49 Headunit
50 Steuergerät Sprachbedienung
51 TV-Tuner MOST
52 Soundverstärker
53 Navigationsrechner
54 Kommunikationsplattform (CP1)
PRIVATE-BUS
5 Sitzsteuergerät Fahrer
6 Sitzsteuergerät Beifahrer
7 Sitzsteuergerät hinten links
8 Sitzsteuergerät hinten rechts
23 TV-Tuner CAN
20 Dachinstrument
45 Sensotronic Brake System (FSG)
47 Sensotronic Brake System (ASG1)
48 Sensotronic Brake System (ASG 2)
55 Multikonturlehne vorne links
56 Multikonturlehne vorne rechts
57 Multikonturlehne hinten links
58 Multikonturlehne hinten rechts
59 Keyless Go Heckmodul
60 Keyless Go Innenraummodul
61 Keyless Go Tür hinten links
62 Keyless Go Tür hinten rechts
63 Fondbildschirm links
64 Fondbildschirm rechts
65 Kommunikationsplattform Fond (CP2)
66 Surround Amplifier
67 Audio Video Controller
68 CD-Wechsler
71 DVD Spieler
72 Sender Kopfhörer links
73 Sender Kopfhörer rechts
KON. VERKABELUNG
38 DC/AC Wandler Elektrotransparentes Panoramadach
39 DC/AC Wandler Elektrotransparente Trennwand
69 Steuergerät Mikrofonarray
70 Gegensprechanlage
74 DC/AC Wandler Leuchtstreifen Dach
75 DC/AC Wandler Elektrolumineszenz Schiebehimmel
76 Spannungswandler DC/DC Solardach
Σ aller Steuergeräte: 76
MAYBACH

Modern cars are computer networks on wheels

So, why is software still so unreliable?

Software must be of high quality: correct, safe & secure

Software must be developed with low effort: cheap & quickly

products include increasing amounts of software

shouldn't get more expensive due to software

releases shouldn't be delayed due to software

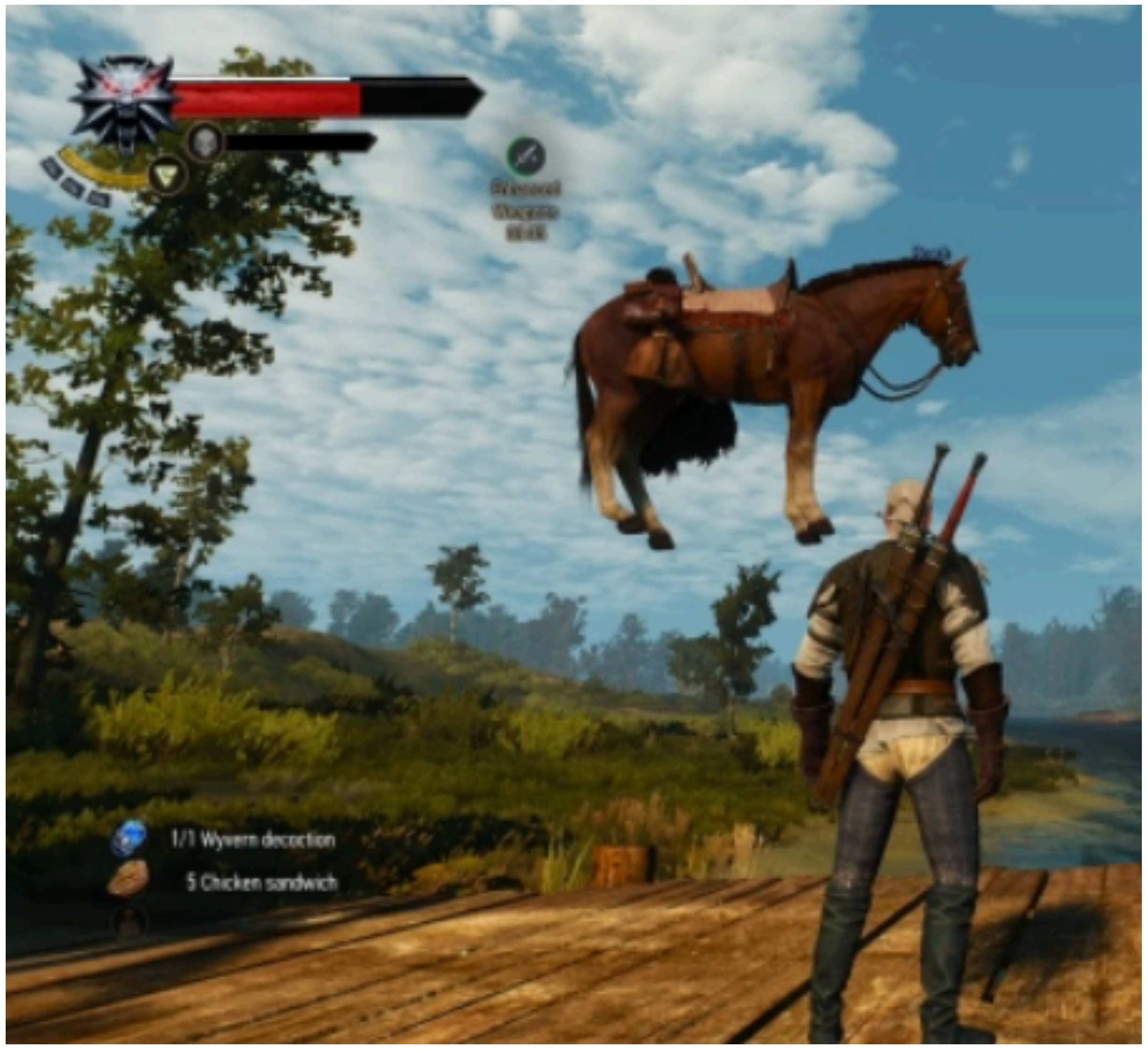
Software defects affect many activities

increasingly used in safety and security critical applications

Financial success often depends on software success

Sometimes we can sacrifice one for the other

- Computer games: development effort is key
  - Correctness and safety are secondary
  - Nobody notices a few pixel with the wrong colour
  - or if the game physics isn't quite accurate….



Sometimes we can sacrifice one for the other

- Flight management system (correctness is key)
  - Safety and security is an overriding concern
  - Defects are very costly and may harm human life

New York Times:

Airline Blames Bad Software in San Francisco Crash

By MATTHEW L. WALD MARCH 31, 2014

Usually, we need a balance

- Consider testing for a mass market product
    - If we test for a very long time with many testers
        - the product will be expensive and we ship too late, but with few defects
    - If we do very little testing
        - the product will be cheaper and ship early, but be riddled with bugs
- In practice, you want to ship when the remaining bugs are unlikely to affect users (sales?) very much

Implications

1. We need to be able to trade quality for reduced effort

    - To be broadly applicable, an approach to software design and implementation must support this trade off

2. We ideally want to increase quality, while reducing effort

    - We seek novel approaches that solve both problems at once

Produce better software with less effort

- Better software
    - Software that has fewer defects (including security defects)
    - Software that is more usable — we won't talk about usability in this course
- Less effort
    - Shorter development time
    - Fewer programmers
    - Less-specialised programmers

The core theme of this course

How can mathematical tools & concepts help to produce better software with less effort?

Why are mathematical tools interesting?

- Mature engineering disciplines are based on mathematical foundations
    - We rarely *guess* the input current to a semiconductor
    - Would you build a bridge and then *test* whether it can hold its load?
- They enable a qualitative change in software development
    - We want to increase quality, while reducing effort
    - High-assurance software requires proof

What mathematical tools?

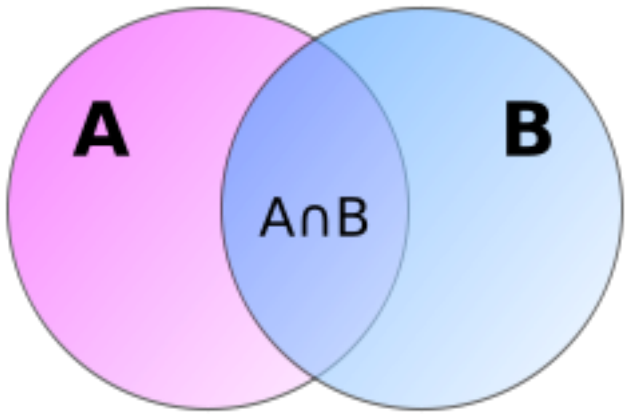
- Tools to reason about specifications and programs
    - Theorem provers & proof assistants
    - Static analysis tools & type checkers
- Tools to transform and refine programs
    - Meta programming & generative programming
    - Rewriting tools
- Advanced programming languages and environments

What is the basis of these tools?

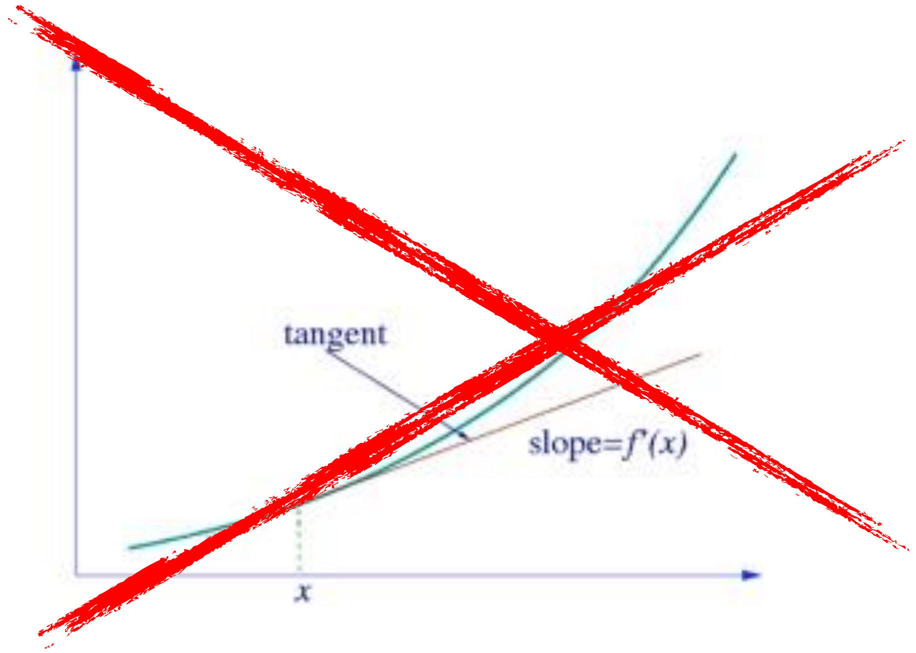
The basis for reasoning about programs

$$\forall x. P(x)$$

Mathematical logic



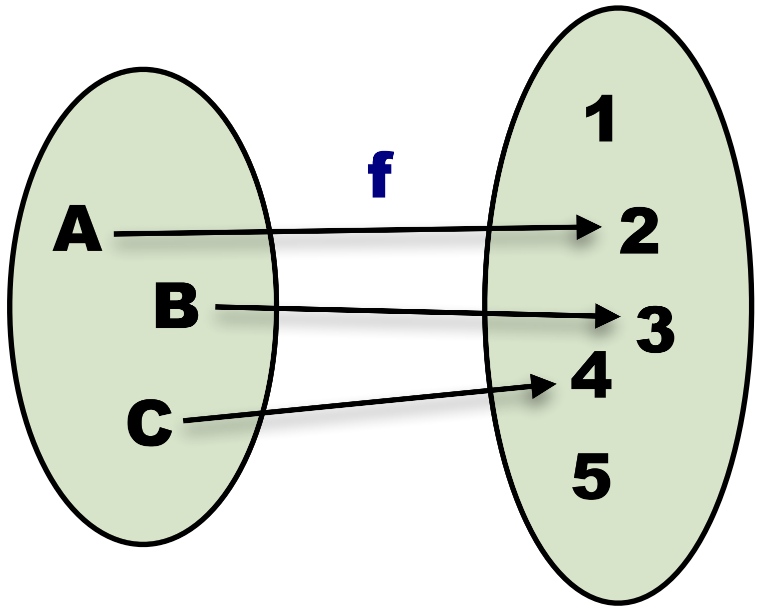
Discrete mathematics



Calculus

Discrete mathematics

- Study of structures that are fundamentally discrete rather than continuous
- In a discrete space, individual points are isolated from each other — consider, natural numbers versus real numbers



functions in computing

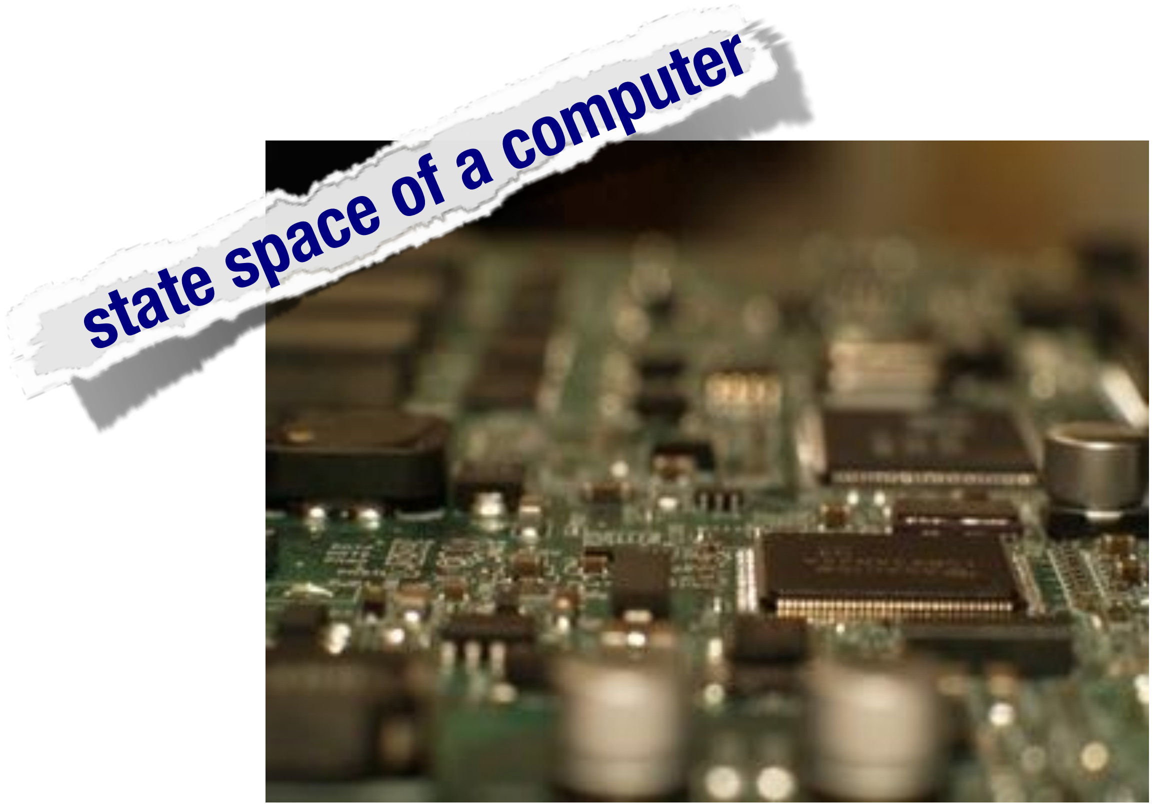


Photo by Un ragazzo chiamato Bi - http://flic.kr/p/4NFrtx

Mathematical logic

- Foundation of reasoning about computation, languages, and programs
    - Mathematical basis for software design and implementation
- Originally arose out of the study of the foundations of mathematics
    - On what ultimate basis can mathematical statements be called true?
    - The foundations of mathematics were heavily disputed in the early 20th century - mainly intuitionists vs formalist

A little bit of history

- 1900: Hilbert's Problems
  - (Initially) ten open problems of mathematics
  - 2nd problem:

    Prove that the axioms of arithmetic are consistent

- 1920s: Hilbert's program
  - Base all of mathematics on a finite,
  - Show that they are complete, cons
- 1928: Hilbert's Entscheidungsproblem (dec
  - given a statement in first order logic, is the statement provable?

$0 \in \mathbb{N}$

$x \in \mathbb{N} \Rightarrow s(x) \in \mathbb{N}$

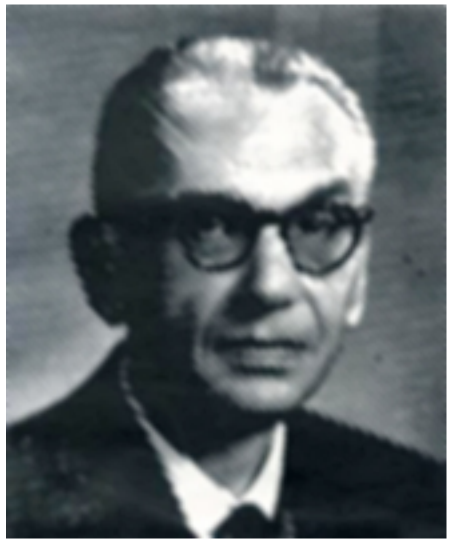
$x \in \mathbb{N} \Rightarrow s(x) \neq 0$

$s(x) = s(y) \Rightarrow x = y$

$0 \in \mathbb{X}, \ (x \in \mathbb{N} \Rightarrow (x \in \mathbb{X} \Rightarrow s(x) \in \mathbb{X})) \Rightarrow \mathbb{N} \subseteq \mathbb{X}$

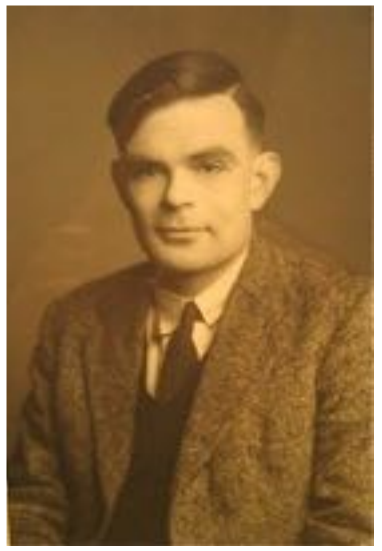
A little bit of history

- 1931: Gödel's incompleteness theorems
  - Consistent, computable axiom set, covering arithmetic, can never be complete
  - Such a system can't even prove it's own consistency



- Adapted form of Hilbert's program
    - Instead of formalising all of mathematics, formalise the important parts
- There are many consistent and complete logics
    - Gödel showed that first-order logic is complete

- 1928: Hilbert's Entscheidungsproblem ("decision problem")
  - Is there an algorithm that, given a mathematical statement (in a formal language), will tell us whether the statement is true or false?
- Church (1936) & Turing's (1937) Church-Turing Theorem
  - Showed that such an algorithm is impossible



- Church-Turing Thesis
  - Recursive functions, Turing machines & lambda calculus are equivalent

Halting Problem

- Idea: assume we have a total function

```
stops (p, i): returns True if program p applied to
input i stops, false otherwise
```

```
f (i) = if stops (i ,i)
          then loop forever
          else 1
```

```
f (f) ?
```

Lambda calculus

- Minimal calculus that can express all computable functions
  - Only variables, function abstraction, and function application
- The basis for many theorem provers
- The foundation of all functional programming languages
  - Extend the lambda calculus with explicit data structures
  - Add syntactic sugar to make it more convenient

Background

- Very simple, but Turing-complete (Church-Turing thesis)
    - Pure lambda calculus has three constructs and two rewrite rules
    - It's compositional (i.e., it's highly modular)
- Today there are many different flavours
    - With and without types, with many extensions, and so on
- Lambda abstractions in C#, C++1x, Objective-C (as blocks), Swift & FP languages

Syntax

- The pure λ-calculus has three syntactic forms
  - Variables: $x, y, z, \ldots$
  - Lambda abstraction: $\lambda x. M$
  - Function application: $M\,N$

Reduction rules

- Alpha conversion:
  - We may change the name of any bound variable
  - For example, we may convert $\lambda x.x$ to $\lambda y.y$
  - A bound variable is one named in an enclosing lambda abstraction
  - For example, here x is bound, but y is free: $\lambda x.xy$

Reduction rules

- Beta reduction:
  - We can reduce a term of the form $(\lambda x.M)N$ to $M[N/x]$
  - Where $M[N/x]$ means to replace any x in M by N
  - The latter process is called substitution
  - For example, we have

execution of a lambda term

$$(\lambda x.x)y \longrightarrow y$$

$$(\lambda x.\lambda y.x)(\lambda z.z)vw \longrightarrow (\lambda y.\lambda z.z)vw \longrightarrow (\lambda z.z)w \longrightarrow w$$

Church Encoding

- How can simple data types and operations on them be encoded in the lambda calculus
  - boolean values?
  - natural numbers?

Course contents

- Logical program properties
  - Basic logic & formal properties
  - Property-based testing
- Types help to design programs
  - Program properties as types
  - Types guide the design
  - Encapsulating properties
- Types help to implement programs
  - Types imply programs
  - Types control effects
  - Types prevent defects
- Effect control helps with parallelism
  - Effects interfere with concurrency & parallelism

Haskell

- A practical, strongly-typed functional programming language
    - Widely used in research, industry & education
    - Mature, highly optimising compiler with interactive environment
    - Over thousands of open-source libraries and tools
- Named after the logician Haskell B. Curry

`http://haskell.org/`

Why Haskell?

- Functional languages are based on the lambda calculus
    - Semantics of programs is fairly precisely defined
    - This simplifies formal reasoning about these programs
- Functional languages can dramatically increase productivity
    - Factor of four has been cited for Erlang versus C++
- Haskell has a very sophisticated type system
- Haskell has controlled effects