Debugging and all that

Software Construction 2012/2013

May 2nd

Jurgen Vinju

Read

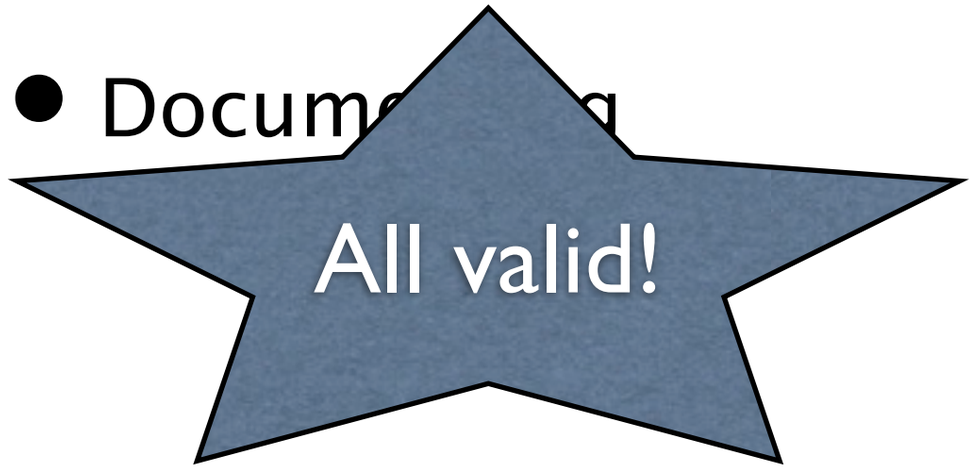
- "The Pragmatic Programmer" (Hunt & Thomas)
- "Why Programs Fail, a guide to systematic debugging" (Zeller)

Today

- Motivating debugging (attitude)
- Terms and concepts (knowledge)
- How to debug (skill)
- Stories (fun)

Programmer Activities

- Learning (API, domain, …)
- Setting up (tools)
- Designing
- Explorative coding
- Exploring code
- Accidental coding (hacking)
- Productive coding
- Testing
- Deploying
- Documenting



Continuous Go/No-Go

- Different best practices with different goals and activities
  - know what you are doing and why and how
  - be able to switch activities and come back
  - step back, realize, plan ahead, take notes
- Go or No-Go: continuously make the technical-debt trade-off
  - quickly estimate cost of activities (in time or in quality)
  - estimate return-on-investment (in time or in quality)
  - estimate available resources (in time)
- Communicate
  - Learn to really listen to what your colleague is saying (and yourself)
  - Learn to explain better and faster what you are thinking

Debugging exists! Plato versus Aristotle

- Wishful thinking and the reality of software
  - Plato: better safe than sorry; “if only we had”
  - Aristotle: wake up in the real world!
- What if you have a bug?
  - Blame somebody else!
  - Blame something else!
  - Give up! Start from scratch!
  - Or… be a **professional programmer**

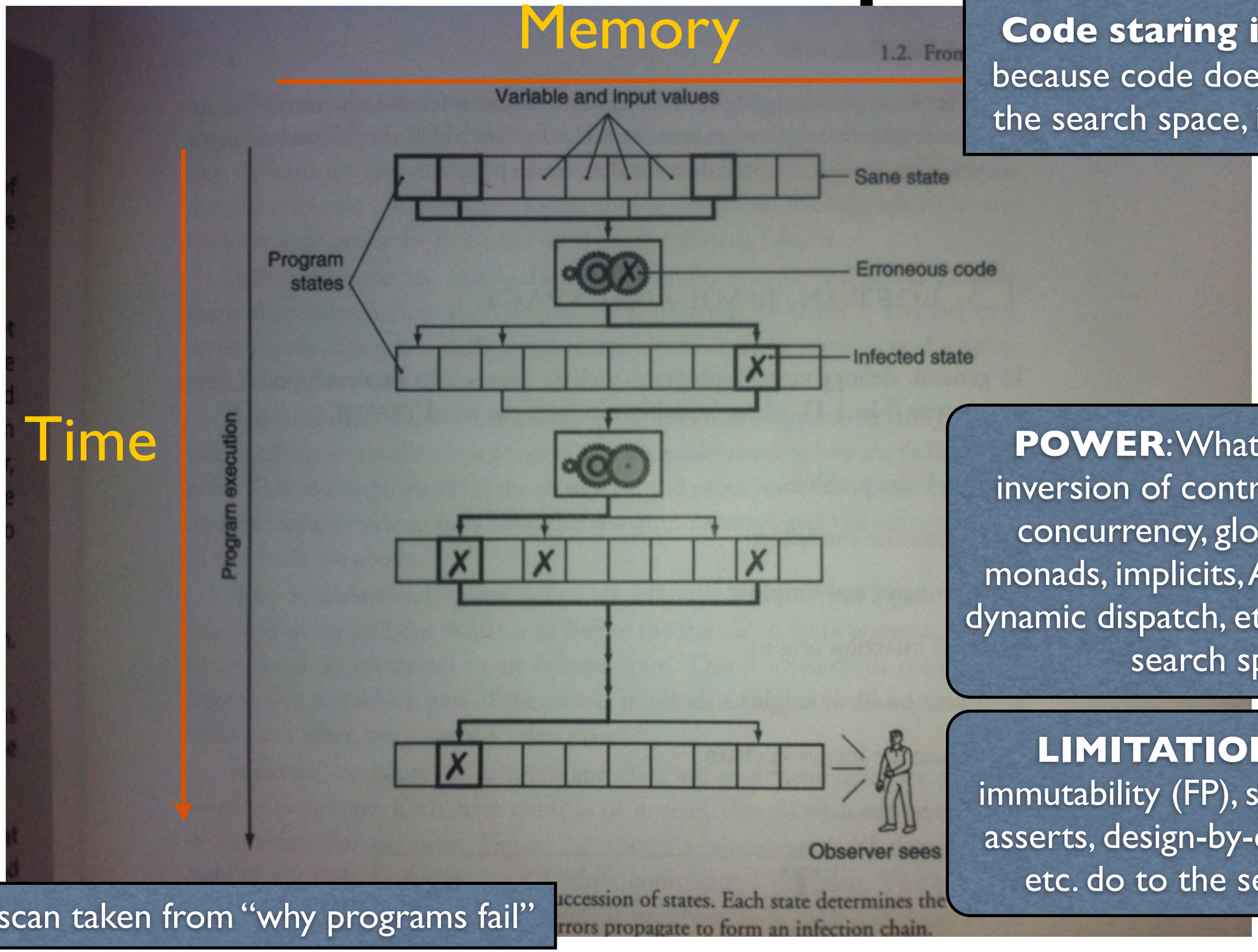
Debugging is search

- Debugging is searching
  - a path from effect (“failure”)
  - to cause (“defect”)
- Preventive (Plato, theoretical)
  - keeping the search space small
- Curative (Aristotle, pragmatic)
  - making the search effective
- Programmers are Researchers

Search space

Memory

Time



Code staring is "useless" because code does not describe the search space, it generates it!

POWER: What do pointers, inversion of control, parallelism, concurrency, global variables, monads, implicits, AOP, interfaces, dynamic dispatch, etc. etc. do to the search space?

LIMITATION: What do immutability (FP), synchronization, asserts, design-by-contract, types, etc. do to the search space?

scan taken from "why programs fail"

First Plato

- Defensive coding
  - express assumptions (in asserts, types, tests)
  - minimize dependencies (locality of debugging)
  - what else?
- Understand why it (should) work!
  - how can you understand a failure if you don't know how to recognize success?
  - understand the relationship between requirements, specification and implementation (bug or feature)
- What else can we do to prevent bugs? Or to make fixing them easier?

Then Aristotle

- The scientific method
  - Observe (actually read the error message...)
  - Document (use an issue tracker; take notes)
  - Reproduce (automate a test)
  - Analyze
  - Simplify
  - Form hypotheses
  - Run a test
  - Fix and see if problem goes away, or go back, without forgetting what you have learned
- Be conscious, and do stuff:
  - What are your (implicit) assumptions/claims/hypotheses?
  - How can you experiment/test/assert that they are true?

Thinking vs Reasoning

- There is a difference
- There are different valid ways of reasoning
  - Deductive (“consequentially”)
  - Inductive (generalizing, “so always/never”)
  - Abductive (guessing, hypothesizing)
- Debugging paradox:
  - The dangerous forms of reasoning are most effective in debugging, making the search space smaller.
  - The safe form of reasoning can get you easily started on wild goose chase.

Deduction

- Direction: from cause to effect
- A deduction guarantees that the conclusion is true given that the premise is true
- “programs only throw segmentation faults if there is a bug in the program” and “this program throws a segmentation fault”, so “this program has a bug

Induction

- Making a general statement after seeing some specific examples
- An induction is/should be made based on many similar observations
- "the program fails every time I pressed the ESC button" (3 times), so "the ESC button must be causing the failure".

Abduction

- Direction: from cause to effect (!)
- Finding an explanation that fits the facts
- Abduction is guessing based on insight
- “The program crashes on my clients machine which runs Windows” and “The program does not crash on my machine which runs Linux”, so: “The cause of the crash is due to

Delta debugging

- How to make a search space smaller?
- Analyze only the differences between what fails and what does not fail
- A definition of “cause”: the minimal difference between a world that shows the effect and a world that does not
- Find the minimal difference, and you have found a cause of the defect.
- Can be iterative, can be automated (Zeller, AskIgor)

Omniscient debugging

- Log EVERYTHING
- Apply delta debugging on the log
- Omniscient debugging tools
  - can automate delta debugging
  - can look back in time, reverse run the program

Live coding

- Debugging is the new programming
- See the effect while you are causing it
- Fixes the forward search, not the backward search

War stories!

- Debugging is a skill of the mind
- Skills are learned by practice and by example
- Let's learn from each other now.

STAR + SNOWBALL

- STAR
  - What was the **S**ituation?
  - What was your **T**ask?
  - How did you **A**pproach?
  - What was het **R**esult?
- SNOWBALL
  - first in pairs of 2
  - then groups of 4, 8, 16
  - each round 5 minutes
  - select the best of two

First listen, then ask, then analyze, then choose

reminiscent?