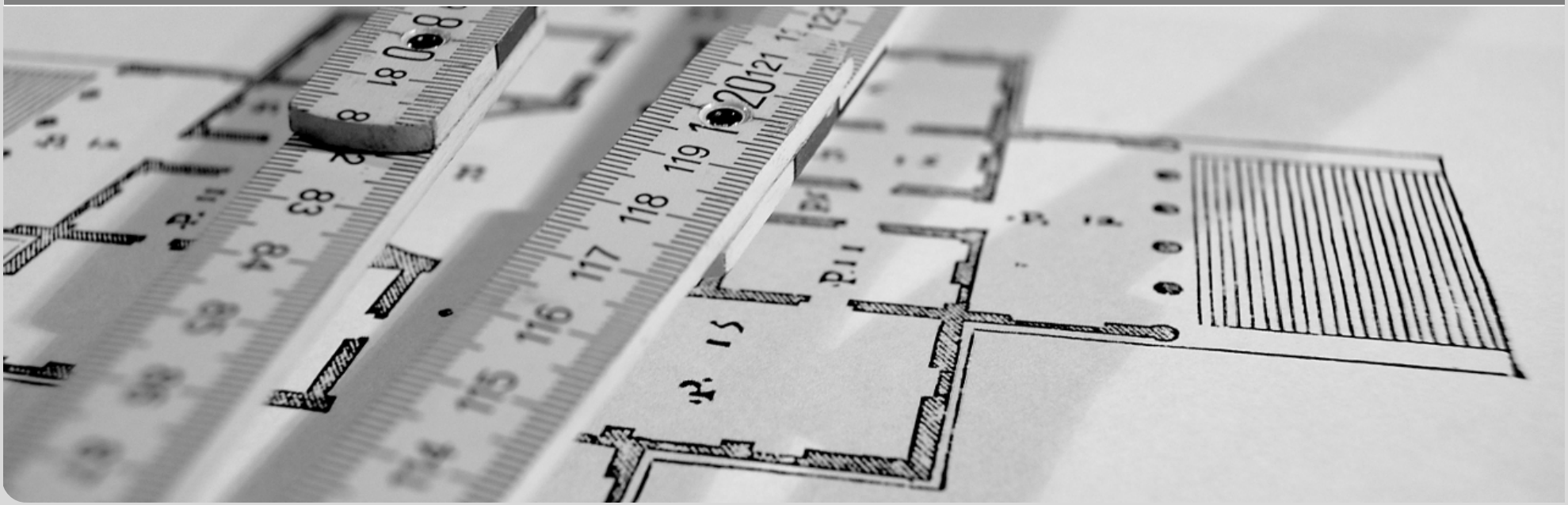
Karlsruher Institut für Technologie

Delta Debugging

A summary of Delta Debugging and its uses

Moritz Laupichler | 19. November 2018

FAKULTÄT FÜR INFORMATIK



Motivation

«Everyone knows that debugging is twice as hard as writing a program in the first place.

So if you're as clever as you can be when you write it, how will you ever debug it?»

– Brian Kernighan in "The Elements of Programming Style"

Version Control allows DD

- Version Control has been around since the 80's
- central terms: configuration, change

The DD idea

Configuration:	Changes:	Configuration:
Yesterday	$\longrightarrow$	Today
	$\longrightarrow$	
Passes tests. ✓	$\longrightarrow$	Tests fail. X

Idea: Delta Debugging

Find the minimal set of changes between Yesterday and Today that induces the failure.

The intuitive approach

- $\mathcal{C} = \{\Delta_1, \ldots, \Delta_n\}$: All changes between Yesterday and Today
- $c \subseteq \mathcal{C}$: A configuration (set of changes applied to Yesterday)
- $test : 2^{\mathcal{C}} \rightarrow \{\checkmark, \times, ?\}$: Result of the tests applied to a configuration

1: **function** SIMPLEDD($c : 2^{\mathcal{C}}$)
2: **if** $|\, c \,| == 1$ **then return** c
3: Split c into two halves c_1, c_2 so that $c_1 \cap c_2 = \emptyset$
4: **if** ($test(c_1) == \times$) **then return** $simpledd(c_1)$
5: **else return** $simpledd(c_2)$

Difficulty #1: Interference

- *ddsimple* works for single failure inducing-changes.
  - In each recursion step it applies only the set of changes known to contain a failure inducing change.
- But what if two changes exist that individually pass the tests but their combination induces failure?

Difficulty #1: Interference

Let $c_1, c_2 \in \mathcal{C}$. c_1 and c_2 **interfere** when $test(c_1) = ✓$, $test(c_2) = ✓$ but $test(c_1 \cup c_2) = ✗$.

Idea: Leave one set of changes applied

If a configuration $c = c_1 \cup c_2$ with $test(c_1) = ✓$ and $test(c_2) = ✓$ is found by *simpledd* then run *simpledd* on c_1 while leaving c_2 applied and vice versa.

Difficulty #2: Inconsistency

The dd+ algorithm

Solves it all!

Some case study

Shows how awesome DD is.

DD as an abstract method

Foundation or part of different solutions.