

Simple crosscutting concerns are not so simple

an AOSD2007 paper

Magiel Bruntink
Arie van Deursen
Maja D'Hondt
Tom Tourwé

Simple crosscutting concerns are not so simple

Simple crosscutting concerns are not so simple

such as tracing

Simple crosscutting concerns **are not so simple**

in terms of the amount of variability in the implementation

Research question

“Is the idioms-based implementation of a crosscutting concern sufficiently systematic such that it is suitable for an aspect-oriented solution”

Tracing @ ASML

- * “Every” function has to trace its arguments
- * Input arguments must be traced at the beginning
- * Output arguments must be traced at the end
- * Approx 16% tracing code in 4 components studied (83 kLoC)

Tracing @ ASML

```
int CC_scan_pending(  
    int scan_id,  
    CC_scan_component scan_component,  
    bool *scan_pending_p) {  
  
    trace("CC", TRACE_INT, func_name,  
        "> (scan_id = %d; scan_component = %s)",  
        scan_id,  
        CC_SCAN_COMPONENT_ENUM2STR(scan_component));  
  
    << BASE FUNCTIONALITY >>  
  
    trace("CC", TRACE_INT, func_name,  
        "< (scan_pending = %b) = %R",  
        *scan_pending_p, r);  
}
```

Basic statistics

	CC1	CC2	CC3	CC4	total
LoC	29339	17848	31165	4985	83337
functions	161	134	174	68	537
parameters types	108	71	65	49	249
tracing macro's	1	1	2	1	2
component names	2	3	1	2	6
function names	3	1	1	1	3

function-level variability

Group all functions that invoke tracing in the same way

```
int CC_scan_pending( ← objects
    int scan_id,
    CC_scan_component scan_component,
    bool *scan_pending_p) {

    trace("CC", TRACE_INT, func_name,
        "> (scan_id = %d; scan_component = %s)",
        scan_id,
        CC_SCAN_COMPONENT_ENUM2STR(scan_component));

    << BASE FUNCTIONALITY >>

    trace("CC", TRACE_INT, func_name,
        "< (scan_pending = %b) = %R",
        *scan_pending_p, r);
}

attributes
```

Extraction from code

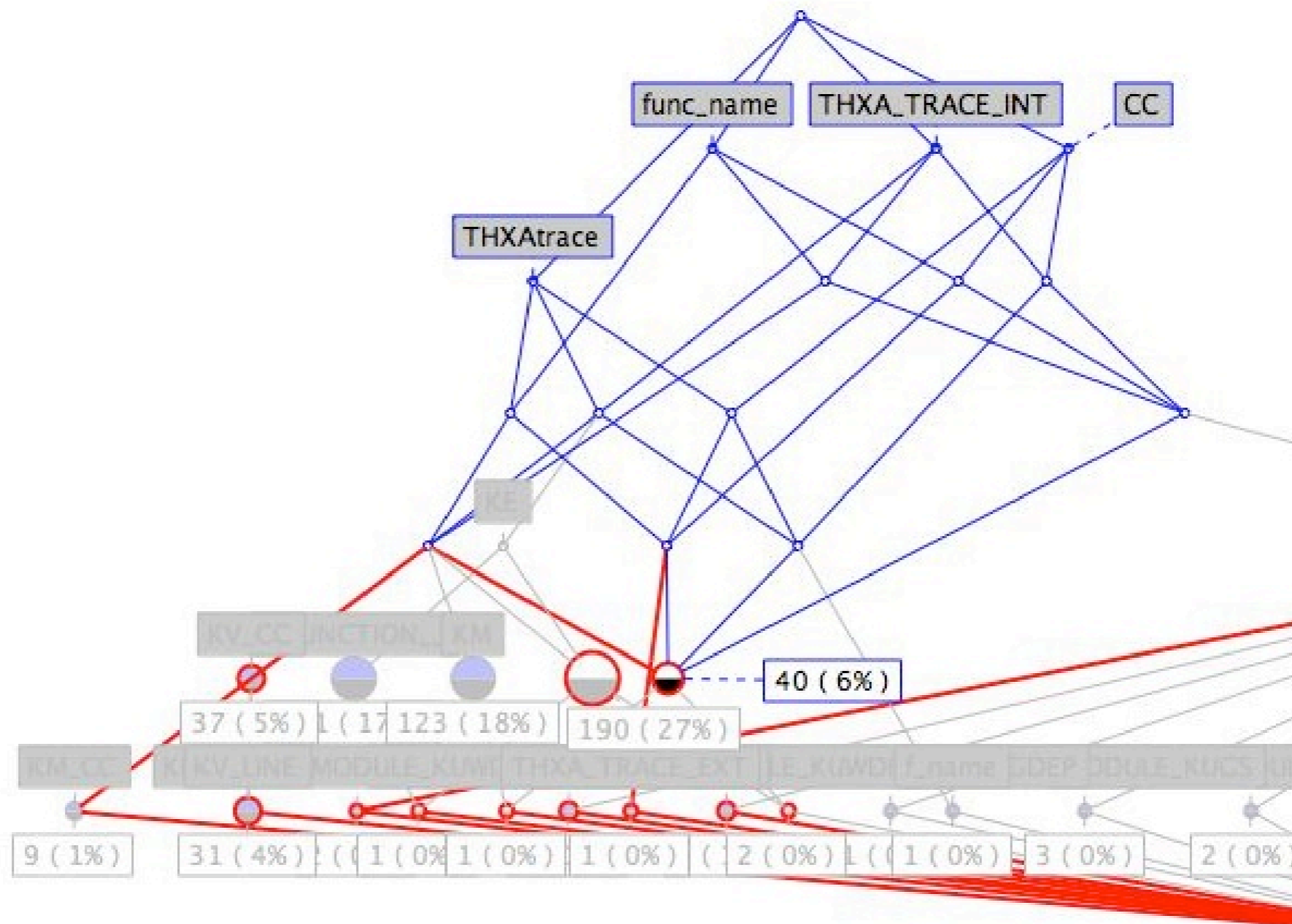
```
set_ill_rema_and_wl : trace "CC1" INT func_name  
set_ill_rema_and_wl : trace "CC1" INT func_name
```

```
get_grating_indices : trace CC INT f_name  
get_grating_indices : trace CC INT f_name
```

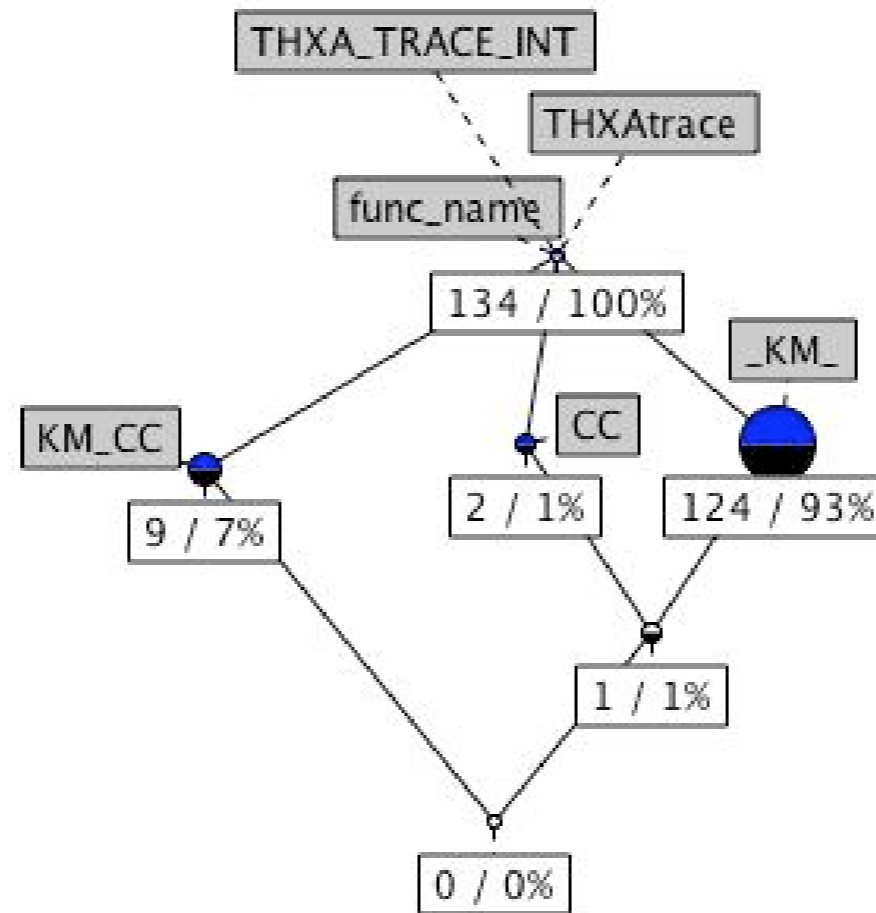
```
CC_rq_PFNRM : trace "CC1" INT func_name  
CC_rq_PFNRM : trace CC INT func_name
```

```
CC_request_agr_height : trace "CC1" INT func_name  
CC_request_agr_height : trace "CC1" INT func_name
```

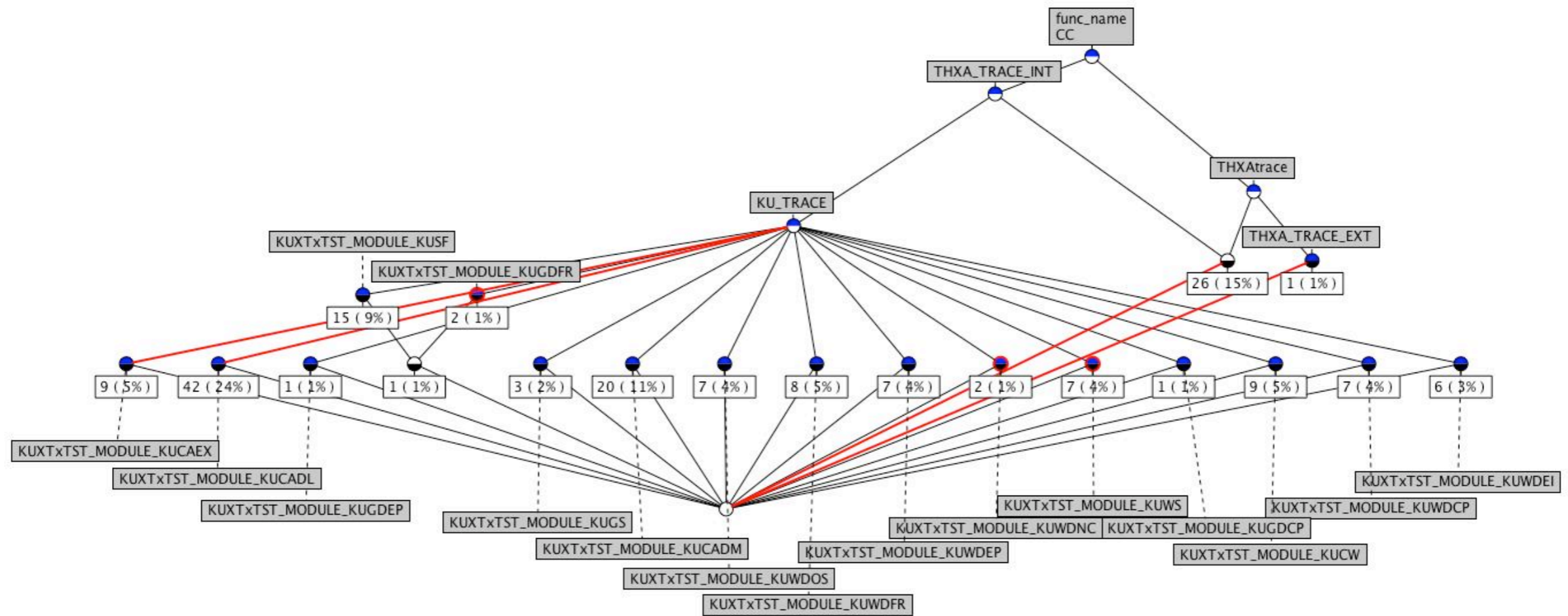
Standard tracing



Inconsistency



Specific tracing variant



Summary

	cc1	cc2	cc3	cc4	total
aspects	6	4	19	2	29
anomalies	2	1	1	0	4

parameter-level variability

Group functions that have a parameter of a certain kind, and that convert that parameter in the same way

```
int CC_scan_pending(  
    int scan_id,  
    CC_scan_component scan_component,  
    bool *scan_pending_p) {  
  
    trace("CC", TRACE_INT, func_name,  
        "> (scan_id = %d; scan_component = %s)",  
        scan_id,  
        CC_SCAN_COMPONENT_ENUM2STR(scan_component));  
  
    << BASE FUNCTIONALITY >>  
  
    trace("CC", TRACE_INT, func_name,  
        "< (scan_pending = %b) = %R",  
        *scan_pending_p, r);  
}
```

Extraction from code

bool: ID

bool: not_traced

REQ_MODE: ID

chuck_id_enum: CHUCK_ID_ENUM2STR()

Summary

	cc1	cc2	cc3	cc4	total
--	-----	-----	-----	-----	-------

inconsistently traced types	11	6	39	8	64
-----------------------------	----	---	----	---	----

consistently traced types	15	5	16	19	55
---------------------------	----	---	----	----	----

Implications

Migration trade-off: textual equivalence vs. solution quality

Textual equivalence: large number of exceptions to generic aspects

Solution quality: possible differences between source and target

Conclusions

- * Migration is feasible
- * Migration trade-off needs attention
- * We have developed technology to help