

# On the effect of scattering and tangling

Tom

# Goal (of the experiment)

Empirically investigate the effect of crosscutting concerns on software quality

effect of crosscutting concerns → scattering and tangling  
software quality → number of faults (in concern code)

# Goal (of the experiment)

Empirically investigate the effect of crosscutting concerns on software quality

effect of crosscutting concerns → scattering and tangling  
software quality → number of faults (in concern code)

→ We have 15.000.000 lines of C code

→ We know exactly which concerns are implemented

# Goal (of the talk)

Please help me to get this experiment right  
(scientifically speaking)

# Goal (of the talk)

Please help me to get this experiment right  
(scientifically speaking)

statistics was never my strongest point ...

# Goal (of the talk)

Please help me to get this experiment right  
(scientifically speaking)

statistics was never my strongest point ...

... and the same holds for empirical experiments

# Hypotheses

$H_{0,s}$ : scattering does not have an effect on the amount of faults in a concern

$H_{0,t}$ : tangling does not have an effect on the amount of faults in a concern

→ try to reject  $H_{0,s}$  and/or  $H_{0,t}$

# Methodology

1. Define **metrics** to measure concern scattering and tangling, and measure it
2. Define **concern verifiers** and measure amount of faults in concern code
3. Determine whether both measures **correlate**, and with what probability

# Definition of "concern"

A concern is a particular piece of interest in a software system

=

$$C = \{ x \mid x \in \text{PDG}, \forall \text{PDG} \in \text{PDGs}(\text{SS}) \}$$

# Considered concerns

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Scatterring

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXATrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXATrace("< result: %d", result);
    return result;
}
```

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Scatterring

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

1. Define metrics
2. Define concern verifiers
3. Determine correlation

# Scattering

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    THXAttrace("< result: %d", result);
    return result;
}
```

scattering(C)

=

$$\#\{x \mid x = \text{pdg}(y), y \in C\} / \#\text{PDGs}(SS)$$

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    THXAttrace("< result: %d", result);
    return result;
}
```

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = CC_PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    THXAttrace("< result: %d", result);
    return result;
}
```

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 1

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 2

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 3

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 4

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 5

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 6

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 7

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (1)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

#transition points = 8

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalng (1)

```

int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_CHECKING;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}

```

tangling(F)  
=

#transition points(F) / #pdg-vertices(F)

#transition points = 8

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (2)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (2)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

external  
control/data  
dependencies

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (2)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

external  
control/data  
dependencies

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (2)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_ERROR;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

external  
control/data  
dependencies

core

tracing

error linking

parameter checking

1. Define **metrics**
2. Define concern verifiers
3. Determine correlation

# Tgnalrig (2)

```
int function1(void* context_ptr, void* data_ptr)
{
    int result = OK;
    THXAttrace("> data_ptr->scan_id: %d", data_ptr->scan_id);
    if((result == OK) && (context_ptr == (void *) NULL)) {
        result = PARAMETER_CHECKING;
        LOG(result, 0, ("parameter 'context_ptr' is null"));
    }
    if(result == OK) {
        result = function2(context_ptr);
        ...
    }
    ...
    THXAttrace("< result: %d", result);
    return result;
}
```

tangling(F)  
=

#control-data dependencies(F) / #pdg-vertices(F)

core

tracing

error linking

parameter checking

1. Define metrics
2. Define concern verifiers
3. Determine correlation

# Fault detection

Based on state machines & CFG traversal

tracing -> determine parameters that are not traced

parameter checking -> determine parameters that are not checked

error linking -> determine incorrect linking, overwriting of error values, etc ...

See our ICSM'05 and ICSE'06 papers and previous talks

1. Define metrics
2. Define concern verifiers
3. Determine correlation

# Spearman's rank-order coefficient

A measure of association between two variables

$$r_s = 1 - \frac{6(\sum D^2)}{N(N^2 - 1)}$$

N = number of subjects

D = difference between a subjects ranks  
on the two variables

$r > 0$  -> variables positively correlated

$r < 0$  -> variables negatively correlated

$r = 0$  -> variables not correlated

1. Define metrics
2. Define concern verifiers
3. Determine correlation

# t-statistic

A measure indicating the probability that the observed value of  $r$  is a chance event

$$t = r_s \sqrt{\frac{N - 2}{1 - r_s^2}}$$

t low → reject H0  
t high → accept H0

# Experiment

	scattering	tangling(1)	tangling(2)	other?
scattering	1			
tangling(1)	x	1		
tangling(2)	y	z	1	
other?	a	b	c	1

# Experiment

	scattering	tangling(1)	tangling(2)	other?
#faults	r1	r2	r3	r4
	t1	t2	t3	t4

# Threats to validity

Our metrics are not the right ones

What is the definition of scattering/tangling then, however?

The observed effect is caused by some other phenomenon (related to scattering/tangling)

For example, the inherent complexity of the concern itself.

We measure concern related faults

What about faults in the core concern? Do faults lead to real bugs?

Your comments?