# Isolating Crosscutting Concerns in Embedded Systems

Arie van Deursen

Joint work with
Magiel Bruntink and Tom Tourwé

## Example Concerns (I)

a) *"Every public function should check its parameters before using them. ..."*

b) *"Every public function should trace itself just after entry and just before exit. ..."*

c) *"Every C function should return an int that is either OK or an error code..."*

d) *"After invoking any C function, the error code returned must be checked..."*

e) *"Functions from layer L1 should wrap error codes from lower layers into L1 error codes"*



## Example Concerns (II)

a) *"The user should be able to undo any command"*

b) *"All Figures should implement the Storable interface"*

c) *"Every figure should notify its observers about any change made"*

d) *"Every component should offer the Lifecycle interface to start or stop execution."*

e) *"Every component should offer an invoke method to allow pipelining a series of commands"*



## Characteristics

- **Quantification**:
  - Do something for selected behavior, functions, classes, methods, layers, exceptions, ...
- **Obliviousness:**
  - Can we decompose the system so that the affected design elements need not be aware of this?

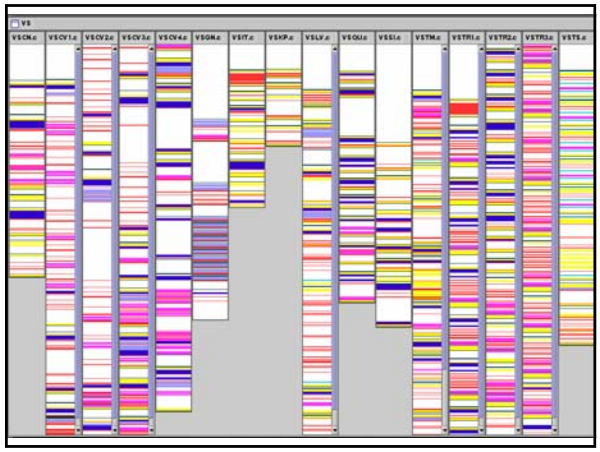
Filman and Friedman, WASC 2000,
*Aspect-Oriented Programming is Quantification and Obliviousness.*



## Composition Risks

- **Crosscutting Concern**
  - Concern that cannot be isolated in a single module but which affects several modules.
- **Scattering**
  - Single concern ("undo", "error handling") scattered across many different components
- **Tangling**
  - Implementation of your primary concern ("move wafer") necessarily tangled with several crosscutting concerns

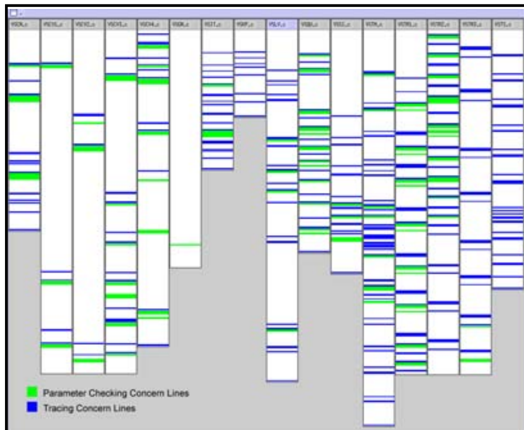


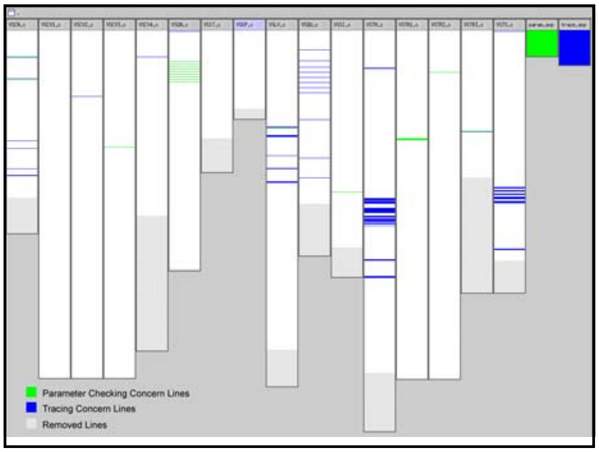


# The Promise of Aspects

- Composable crosscutting concerns
- Static quantification:
  - Extend classes with method *m*
- Dynamic quantification
  - Execute *advice* at explicitly specified execution points (“joinpoints”)
- Weave relevant code into application



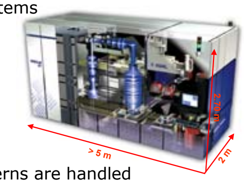



# The Ideals Project

Idiom Design for Embedded App’s on Large Scale

- Analyze 10 Mloc ASML C code base
  - World leader in lithography systems
- Objectives
  - Effort reduction
  - Lead time reduction
  - (Reliability improvement)
- Means
  - Address way crosscutting concerns are handled
- Partners: ASML, ESI, UT, TU/e, CWI, TUD






# Approach

- Analyze present situation
  - *Are there any problems?*
- Envision target situation
  - *Are aspects the solution?*
- Analyze migration issues
  - *Can we get there for existing components?*



# Present Situation

- Language used is C
  - No support for crosscutting concerns
- Solution
  - Coding conventions / ***idioms***
  - Prescribed in architecture documents for most important crosscutting concerns
  - No fixed approach for other crosscutting concerns

## Example I: The *Parameter Checking* Concern

- Each public function should check its parameters
  - Reduce risk of null pointers errors at integration time
- Check should occur before first use / dereference
  - But can be anywhere in the call chain
- Check depends on parameter's kind:
  - input, output, output pointer.

CWI 13 TUDelft SWERL

## Example Code

```
int align (int* data_ptr) {
    int error = OK;
    char* func_name = "align";

    if ((error == OK) && (data_ptr == (int*) NULL))
    {
        error = PARAMETER_ERR;

        LOG(error, 0,
           ("%s: input parameter '%s' is NULL.",
            func_name,
            "data_ptr"));
    }
    . . .
```

CWI 14 TUDelft SWERL

## Code Size Implications

- CC : component for data conversion
- 19 KLOC, 11 files.

| Concern | LOC | % |
|---|---|---|
| Tracing | 1539 | 8% |
| Error Handling | 1716 | 9% |
| Memory Handling | 965 | 5% |
| Parameter Checking | 1441 | 7% |
| Total | 5561 | 29% |

CWI 15 TUDelft SWERL

## Quality Implications

- Parameter checking fault model:
  - Check omitted
  - Wrong check, confusing in- / output
  - Unnecessary check
- Develop static checker for correct usage of idiom
  - Data & control flow analysis
  - *Plugin* to CodeSurfer program analysis tool

CWI 16 TUDelft SWERL

## Parameter Checking Findings

| | Size Kloc | Deviations reported | Unintended | Intended |
|---|---|---|---|---|
| **CC1** | 3 | 8 | 0 | 8 |
| **CC2** | 19 | 65 | 58 | 7 |
| **CC3a** | 12 | 23 | 16 | 7 |
| **CC3b** | 98 | 53 | *41* | *12* |
| **CC4** | 14.5 | 31 | *24* | *7* |
| **CC5** | 15 | 5 | *4* | *1* |
| | **161.5** | **185** | ***143 (77%)*** | ***42 (23%)*** |

0.9 missing checks per KLOC
(1 in 4 params not checked)

## Example II The Error Linking Concern

- Proper error logging essential for
  - Activating correct recovery procedure
  - Making sure repair engineer can take proper action upon crash
  - Directly affects *repair time,* hence *uptime.*
- Different error codes are *linked* back to *root cause*
- Well described error linking idiom

CWI 18 TUDelft SWERL

# Error Linking

```
int queue_add(CCQU_queue *queue, void *item_data, bool front) {
  int  r = OK;
  …
  if ((r == OK) && (queue == (CCQU_queue *) NULL)) {
     r = CCXA_PARAMETER_ERR;
     ERXA_LOG(r, 0);
  }
  …
  if (r == OK) {
     r = PLXAmem_malloc(sizeof(CCQU_queue_item), (void **) &qi);
     if (r != OK) {
        ERXA_LOG(r, 0);
        ERXA_LOG(CCXA_MEMORY_ERR, r);
        r = CCXA_MEMORY_ERR;
     }
  }

  if (r == OK) {
     ...
  }

  return r;
}
```

Initialization

Root error

Error logging

Skipping

Linking

Propagation

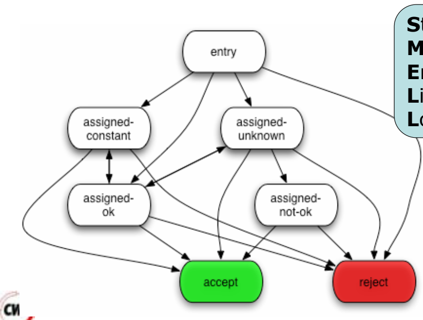
# Error Linking Quality

- Fault model:
  - Wrong error variable returned,
  - assigned and logged value mismatch,
  - not linked to previous value,
  - unsafe assignment, …
- Can be identified using analysis of *program dependence graph*
- For each path find out whether error value is properly set, checked, logged and returned

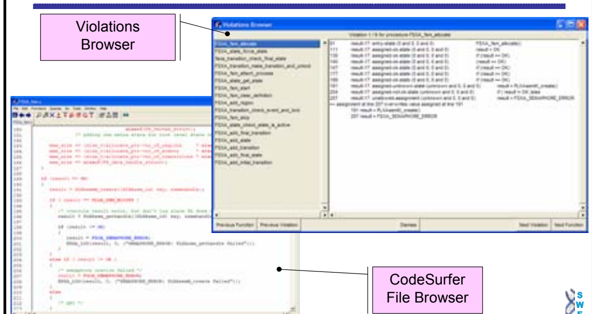


# Static Analysis in SMELL

**S**tate **M**achine for **E**rror **L**inking and **L**ogging




# Violation reports






# SMELL 0.2 Initial Findings

| | KLOC | Reported Deviations | False Positives | Limitations | Validated Deviations |
|---|---|---|---|---|---|
| CC1 | 3 | 32 | 2 | 4 | 26 |
| CC2 | 19 | 72 | 20 | 24 | 28 |
| CC6 | 15 | 16 | 0 | 3 | 13 |
| CC4 | 14.5 | 107 | 14 | 13 | 80 |
| CC5 | 15 | 9 | 0 | 3 | 6 |
| | 66.5 | 236 | 36 | 47 | 153 |

15% 20% ~2/Kloc



# Crosscutting Concerns: Current Situation

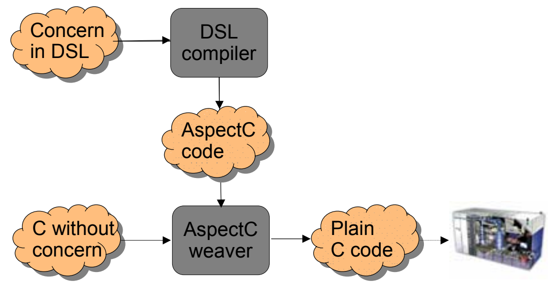
- Preliminary results
- 30% of code devoted to boiler plate code
- 3 deviations from idiom per KLOC
  - Just for parameter checking and error linking

## Envisioned Solution

- Domain-specific aspect-oriented language
  - Can be separate from base C code
  - Can also be integrated by means of annotations
  - PCSL: Para Checking Spec Language
- Weaver needed to inject advice at proper places

25

## DSL Weaving




## Example PCSL specification

```
component CC {

    int align (input int* data_ptr);

    <other signatures>

    input advice {
        if(thisParameter.name == (thisParameter.type) NULL) {
            error = PARAMETER_ERR;
            LOG(r,0,("%s: Input parameter %s error (NULL)",
                      thisParameter.function.name.asString,
                      thisParameter.name.asString));
        }
    }

    <advices for other parameter kinds>
}
```

## Results

| Parameter Checking LOC | Original | PCSL |
|---|---|---|
| CC1 3KLOC | 56 | 46 |
| CC2 19KLOC | 961 | 132 |
| CC3 110KLOC | 456 | 787 |

28

## PCSL Findings

Pro
- Better documentation
- Better checks
- Maint. & dev. effort reduced
- Cleaner and less code

Con
- Dev. effort replaced by tool dev. effort
- Non-standard technology
- Adoption / migration low risk, but not zero risk

29

## Error Linking Alternatives

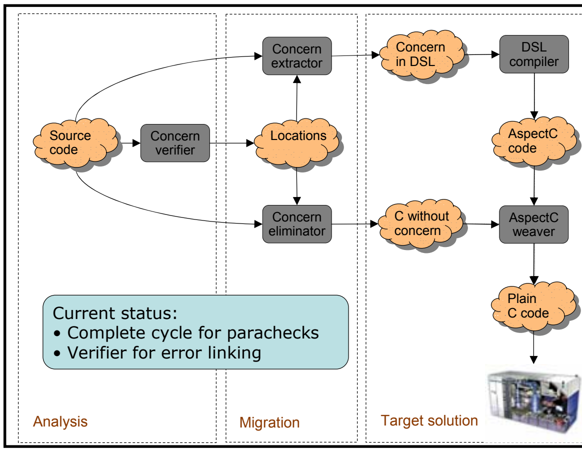
- Thinking with ASML about alternative macrosets
  - Avoid most likely faults as we identified them
  - Closer to "try-catch" mechanism
  - Consider relying on setjmp / longjmp
- Step 2: Further improvements using aspect weaving

Work in Progress!

# Migration Path

- Incremental adoption for existing components
  - At function, module, component, or subsystem level
- **Concern verification:**
  - Determine locations for concern (= SMELL)
- **Concern extraction:**
  - Recover, e.g., PCSL specification
- **Concern elimination:**
  - Clean up C code






# Outlook

- Expand parachecking to related concerns
  - tracing, timing
- Expand error linking concern
- Consider idioms in design documents
  - Often not prescribed / implicit
  - Formalize using UML profiles
  - Use model trafos to obtain code



# Composition Implications?

- Is error handling a component?
- Can we compose components that have different error handling requirements?
- Can I impose my error handling regime on 3d party components?
- *<< your question goes here! >>*



# Conclusions

- Crosscutting concerns are unavoidable
- Idiomatic approach may
  - contribute to upto 30% of code size
  - several idiom deviations per kloc
- Domain-specific aspect languages:
  - way to reduce effort and faults
  - migration support essential and feasible
- Investigating opportunities for (design-level) implicit idioms



# More Information


- M. Bruntink, A. van Deursen, and T. Tourwé. *Isolating idiomatic crosscutting concerns.* ICSM 2005. IEEE.
- M. Bruntink, A. van Deursen, R. van Engelen and T. Tourwé. *On the use of clone detection for identifying crosscutting concern code*. IEEE TSE, 2005.
- M. Bruntink, A. van Deursen and T. Tourwé. *Discovering Faults in Idiom-Based Exception Handling*. Submitted, 2005
- M. Marin, A. van Deursen, and L. Moonen. *Identifying aspects using fan-in analysis*. WCRE 2004, IEEE.
- B. Graaf, S. Weber, A. van Deursen. *Migrating Supervisory Machine Control Architectures using Model Transformations.* WICSA 2005, IEEE.