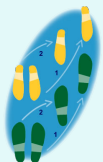


Isolating Crosscutting Concerns in System Software

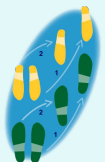
Magiel Bruntink, Arie van Deursen, Tom Tourwé

C.W.I.

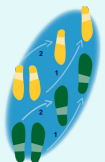
Amsterdam, The Netherlands



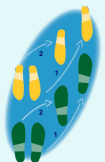
- ☑ Introduction to AOSD
- ☑ Goals of Ideals project & experiment
- ☑ Idioms-based approach
- ☑ Different adoption strategies for migrating toward AOSD approach
- ☑ Observations
- ☑ Conclusions



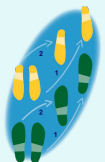
- ☑ Introduction to AOSD
- ☑ Goals of Ideals project & experiment
- ☑ Idioms-based approach
- ☑ Different adoption strategies for migrating toward AOSD approach
- ☑ Observations
- ☑ Conclusions



- ❑ Tyranny of the dominant decomposition
 - “No matter how well a system is decomposed into modular units, some functionality will always crosscut this modularity”
- ❑ Typical examples include logging, tracing, exception handling, transaction management, concurrency, etc ...

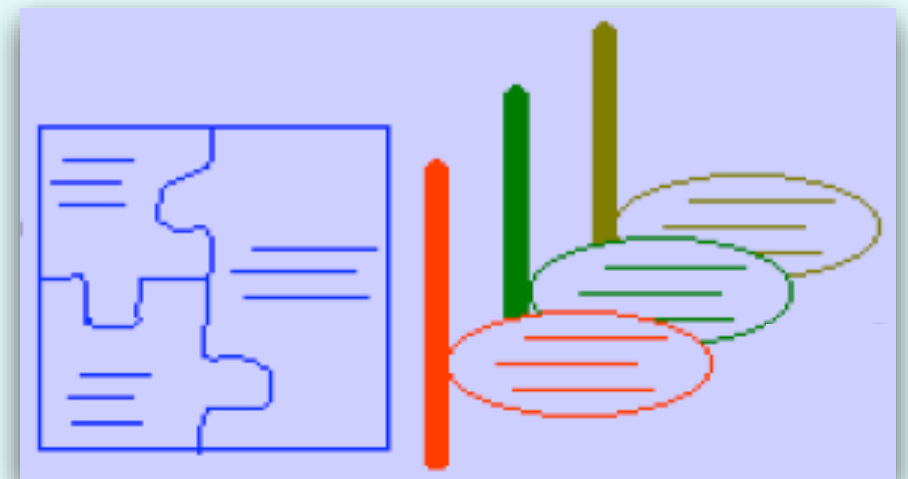


- ☑ Add extra abstraction mechanism ("aspect") on top of existing mechanisms
 - ☑ Captures concern code in one single module -> "advice code"
 - ☑ Concern code removed from ordinary ("base") program
 - ☑ Specifies where code needs to be inserted in the original program -> "pointcuts"
- ☑ Aspect "weaver" combines base code and aspects into executable program

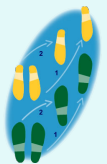




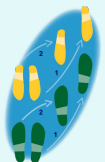
Ordinary application



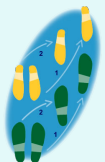
Aspect-oriented application



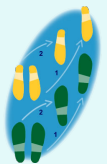
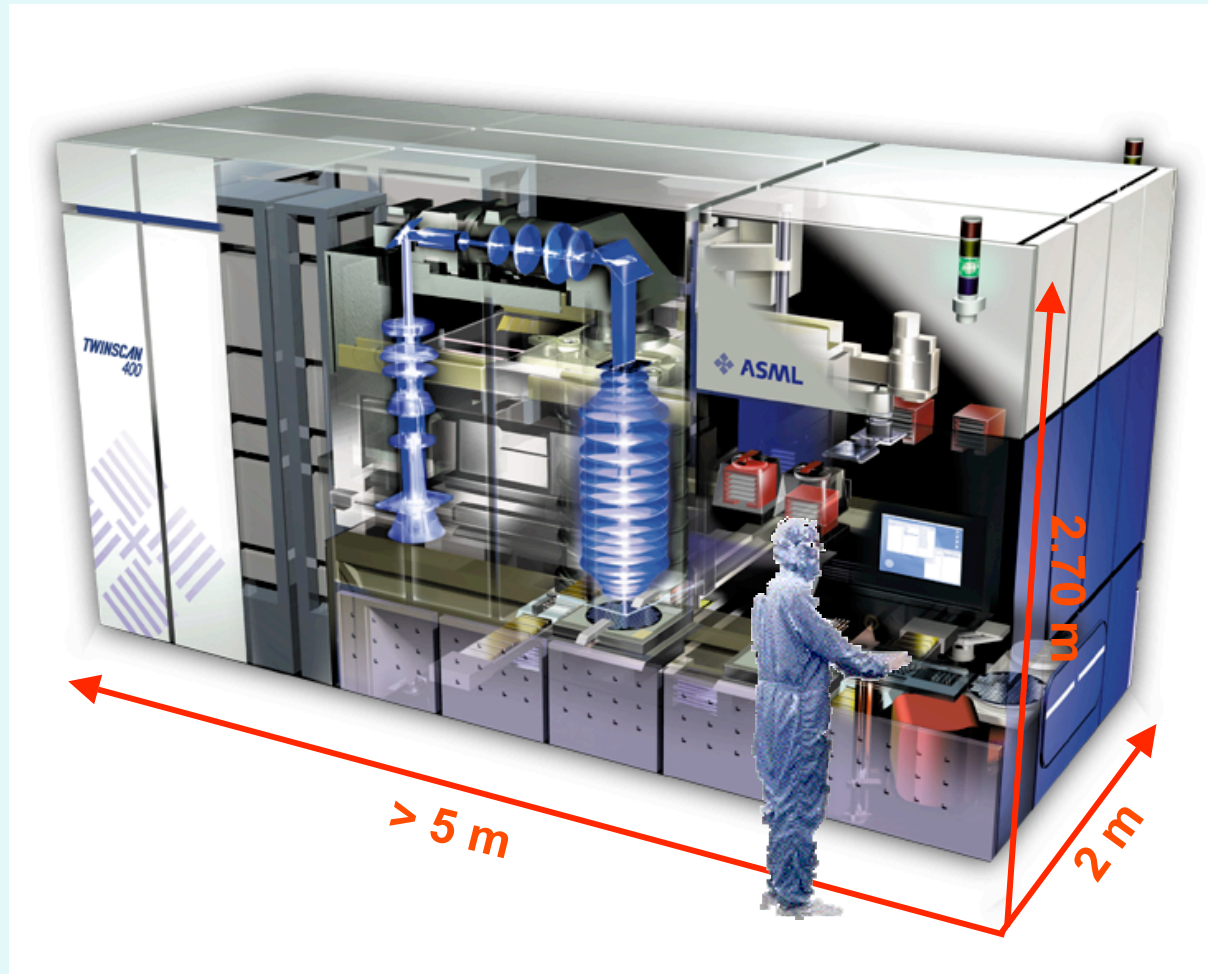
- ☑ Introduction to AOSD
- ☑ Goals of Ideals project & experiment
- ☑ Idioms-based approach
- ☑ Different adoption strategies for migrating toward AOSD approach
- ☑ Observations
- ☑ Conclusions



- ☑ Ideals = **I**diom **D**esign for **E**Embedded
Applications on **L**arge **S**cale
- ☑ Participants
 - ☑ ASML, Embedded Systems Institute (ESI),
Universiteit Twente (UT), Universiteit Eindhoven
(TU/e), Centrum voor Wiskunde & Informatica
(CWI)

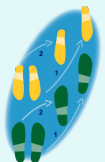


Ideals project



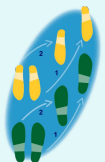
Ideals project

- reduce the development effort
- reduce the errors
- reduce the code size
- improve maintainability and evolvability



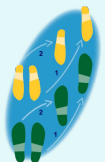
Ideals project

- ☑ Improve the handling of crosscutting concerns in embedded software in order to
 - ☑ reduce the development effort compared to manually implementing the concerns
 - ☑ reduce the errors introduced by manually implementing the concerns
 - ☑ reduce the code size because concern code is no longer duplicated
 - ☑ improve maintainability and evolvability

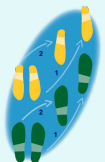


This experiment

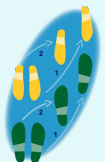
- ☑ Assess feasibility of automatically **migrating idioms-based approach** to crosscutting concerns into **AOSD approach**
- ☑ Evaluate advantages & disadvantages of AOSD approach w.r.t. Ideals goals



- ☑ Introduction to AOSD
- ☑ Goals of Ideals project & experiment
- ☑ **Idioms-based approach**
- ☑ Different adoption strategies for migrating toward AOSD approach
- ☑ Observations
- ☑ Conclusions



- ☑ Based on naming conventions and coding idioms
- ☑ Copy-paste-adapt solution
- ☑ Due to absence of language constructs for expressing crosscutting concerns

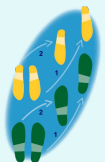


Parameter Checking Concern

```
int CC_queue_empty(CC_queue *queue, bool *empty) {
    ...
    if(queue == (CC_queue *) NULL) {
        r = VSXA_PARAMETER_ERR;
        ERXA_LOG(r,0,("%s: Input parameter %s error (NULL)", "CC_queue_empty", "queue"));
    }
    ...
}

int CC_queue_peek_front(CC_queue *queue, void **queue_data) {
    ...
    if(queue_data != (void **) NULL) {
        r = VSXA_PARAMETER_ERR;
        ERXA_LOG(r,0,("%s: Output pointer parameter %s error (NULL)", "void", "queue_data"));
    }
    ...
}

int CC_queue_init(CC_queue *queue, void *queue_data)
    ...
    if(queue == (CC_queue *) NULL) {
        r = VSXA_PARAMETER_ERR;
        ERXA_LOG(r,0,("%s: Output parameter %s error (NULL)", "CC_queue_empty", "queue"));
    }
    ...
}
```

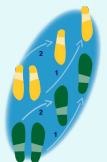


Parameter Checking Concern

```
int CC_queue_empty(CC_queue *queue, bool *empty) {
    ...
    if(queue == (CC_queue *) NULL) {
        r = VSXA_PARAMETER_ERR;
        ERXA_LOG(r,0,("%s: Input parameter %s error (NULL)", "CC_queue_empty", "queue"));
    }
    ...
}

int CC_queue_peek_front(CC_queue *queue, void **queue_data) {
    ...
    if(queue_data != (void **) NULL) {
        r = VSXA_PARAMETER_ERR;
        ERXA_LOG(r,0,("%s: Output pointer parameter %s error (NULL)", "void", "queue_data"));
    }
    ...
}

int CC_queue_init(CC_queue *queue, void *queue_data)
    ...
    if(queue == (CC_queue *) NULL) {
        r = VSXA_PARAMETER_ERR;
        ERXA_LOG(r,0,("%s: Output parameter %s error (NULL)", "CC_queue_empty", "queue"));
    }
    ...
}
```



Parameter Checking Concern

Input

```
int CC_queue_empty(CC_queue *queue, bool *empty) {  
    ...  
    if(queue == (CC_queue *) NULL) {  
        r = VSXA_PARAMETER_ERR;  
        ERXA_LOG(r,0,("%s: Input parameter %s error (NULL)", "CC_queue_empty", "queue"));  
    }  
    ...  
}
```

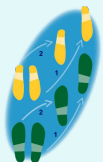
Output

```
int CC_queue_peek_front(CC_queue *queue, void **queue_data) {  
    ...  
    if(queue_data != (void **) NULL) {  
        r = VSXA_PARAMETER_ERR;  
        ERXA_LOG(r,0,("%s: Output pointer parameter %s error (NULL)", "void", "queue_data"));  
    }  
    ...  
}
```

Output
Pointer

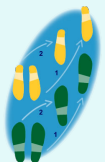
```
int CC_queue_init(CC_queue *queue, void *queue_data)  
    ...  
    if(queue == (CC_queue *) NULL) {  
        r = VSXA_PARAMETER_ERR;  
        ERXA_LOG(r,0,("%s: Output parameter %s error (NULL)", "CC_queue_empty", "queue"));  
    }  
    ...  
}
```

Deviation

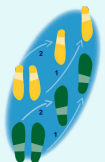


Idioms-based approach

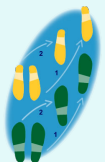
- ☑ Requires effort & discipline
- ☑ Prone to errors
- ☑ Decreases maintainability, evolvability, ...
 - ☑ code duplication
 - ☑ scattering
 - ☑ tangling



- ☑ Introduction to AOSD
- ☑ Goals of Ideals project & experiment
- ☑ Idioms-based approach
- ☑ Different adoption strategies for migrating toward AOSD approach
- ☑ Observations
- ☑ Conclusions

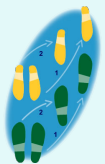


- ☑ separate concern code from key functionality
 - ☑ represent in domain-specific language
- ☑ weave concern code with functional code
 - ☑ reuse existing AspectC weaver
- ☑ analyse & migrate existing ASML source code
 - ☑ Different adoption strategies



1st Strategy

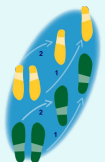
No automated weaving



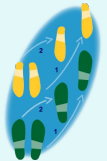


Concern verifier

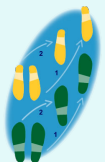
- ☑ identifies deviations & locations
- ☑ works on existing ASML source code
- ☑ implemented as a plugin to CodeSurfer



```
ideals/src/VS/csurf/VSSQU.c
File Edit Functions Queries Go Tools Window Help
VSQU.c VSTR.c VSTS.c VSCN.c VSCU.c
220
221.     return queue_add(queue, item_data, front = TRUE);
222 }
223
224 /*-----*/
225 int VSQU_queue_add_back(VSQU_queue *queue, void *item_data)
226
227 /*
228 * Input(s) : item data
229 *            data field to add to the front of queue q
230 * Output(s) : <none>
231 * InOut(s)  : queue
232 *            queue to add to
233 * Returns   : error status
234 *            - OK
235 *            - VSXA_MEMORY_ERR
236 * Purpose   : Add item to the back of a queue
237 * Notes     :
238 */
239 {
240     bool front;
241
242     return queue_add(queue, item_data, front = FALSE);
243 }
244 /*-----*/
245 int VSQU_queue_peek_front(VSQU_queue *queue, void **item_data)
246
247 /*
248 * Input(s) : queue
249 *            queue to peek at
250 * Output(s) : item data
251 *            data field at the front of queue q
252 * InOut(s)  : <none>
253 * Returns   : error status
254 *            - OK
255 *            - VSXA_PARAMETER_ERR
256 * Purpose   : Examine the item at the front of a queue
257 * Notes     :
258 */
259 {
260     const bool front = TRUE, delete_item = FALSE;
261
```



- ❑ Automatically extracts parameter kinds & parameter checks from the code
- ❑ Requires domain information to separate “intended” from “unintended” deviations
- ❑ Developed iteratively
 - ❑ to reduce false positives to a minimum
 - ❑ to refine the rules the concern should adhere to
 - ❑ with suggestions from domain expert



Experimental Results

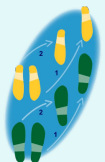
confirmed results
for CC1 (19k LoC)

parameters	checks	deviations	unintended
245	149	110	96

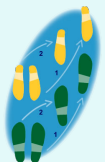
40% of parameters
is not checked in the
appropriate way!

preliminary results
for CC2 (110k LoC)

parameters	checks	deviations	unintended
705	116	284	?

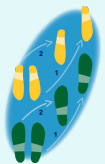


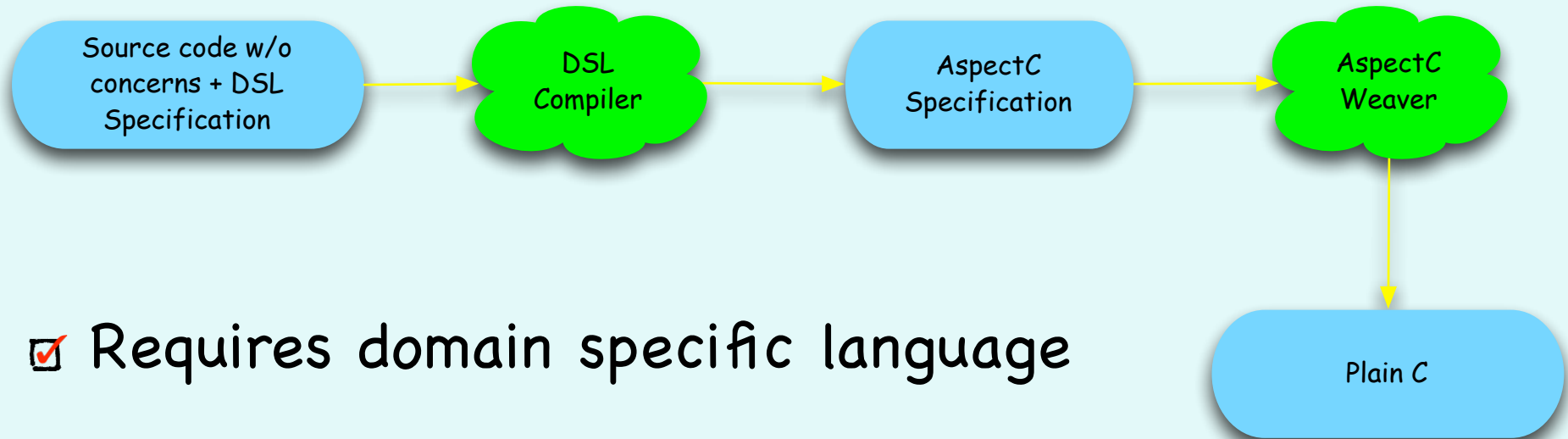
- ☑ Improved code quality
 - ☑ consistency & correctness of parameter checking
- ☑ Guidance for developers
 - ☑ optimal locations for checks can be computed, to reduce the amount of programming effort needed



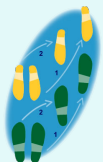
2nd Strategy

Automated weaving for new code only



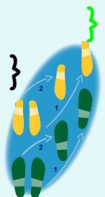


- ☑ Requires domain specific language
- ☑ Reuses existing technology
- ☑ Co-exists with existing code



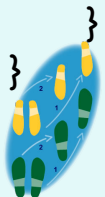
Parameter Checking DSL

```
component CC {
  CC_queue_peek_front(output CC_queue *queue, output output_pointer void **queue_data);
  CC_queue_peek_back(output CC_queue *queue, output output_pointer void **queue_data);
  CC_queue_empty(input CC_queue *queue, output bool *empty);
  CC_queue_init(output CC_queue *queue, deviation void *queue_data);
  ...
  input advice {
    if(thisParameter.name == (thisParameter.type) NULL) {
      r = CC_PARAMETER_ERR;
      CC_LOG(r,0,(%s: "Input parameter %s error (NULL)", thisParameter.function.name, thisParameter.name));
    }
  }
  output advice {
    if(thisParameter.name == (thisParameter.type) NULL) {
      r = CC_PARAMETER_ERR;
      CC_LOG(r,0,(%s: "Output parameter %s error (NULL)", thisParameter.function.name, thisParameter.name));
    }
  }
  output pointer advice {
    if(*thisParameter.name != (thisParameter.type) NULL) {
      r = CC_PARAMETER_ERR;
      CC_LOG(r,0,(%s: "Output parameter %s may already contain a value. This value will be overwritten, which "
        "may lead to a memory leak", thisParameter.function.name, thisParameter.name));
    }
  }
}
```

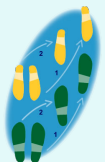


Parameter Checking DSL

```
component CC {
  CC_queue_peek_front(output CC_queue *queue, output output_pointer void **queue_data);
  CC_queue_peek_back(output CC_queue *queue, output output_pointer void **queue_data);
  CC_queue_empty(input CC_queue *queue, output bool *empty);
  CC_queue_init(output CC_queue *queue, deviation void *queue_data);
  ...
  input advice {
    if(thisParameter.name == (thisParameter.type) NULL) {
      r = CC_PARAMETER_ERR;
      CC_LOG(r,0,(%s: "Input parameter %s error (NULL)", thisParameter.function.name, thisParameter.name));
    }
  }
  output advice {
    if(thisParameter.name == (thisParameter.type) NULL) {
      r = CC_PARAMETER_ERR;
      CC_LOG(r,0,(%s: "Output parameter %s error (NULL)", thisParameter.function.name, thisParameter.name));
    }
  }
  output pointer advice {
    if(*thisParameter.name != (thisParameter.type) NULL) {
      r = CC_PARAMETER_ERR;
      CC_LOG(r,0,(%s: "Output parameter %s may already contain a value. This value will be overwritten, which "
        "may lead to a memory leak", thisParameter.function.name, thisParameter.name));
    }
  }
}
```

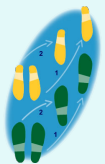


- ☑ Only annotating deviations suffices
 - ☑ parameter kinds can be inferred automatically
- ☑ Advice code/annotations can be separated to ensure consistency of DSL/C code
 - ☑ advice code does not change (often), annotations can change

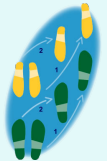
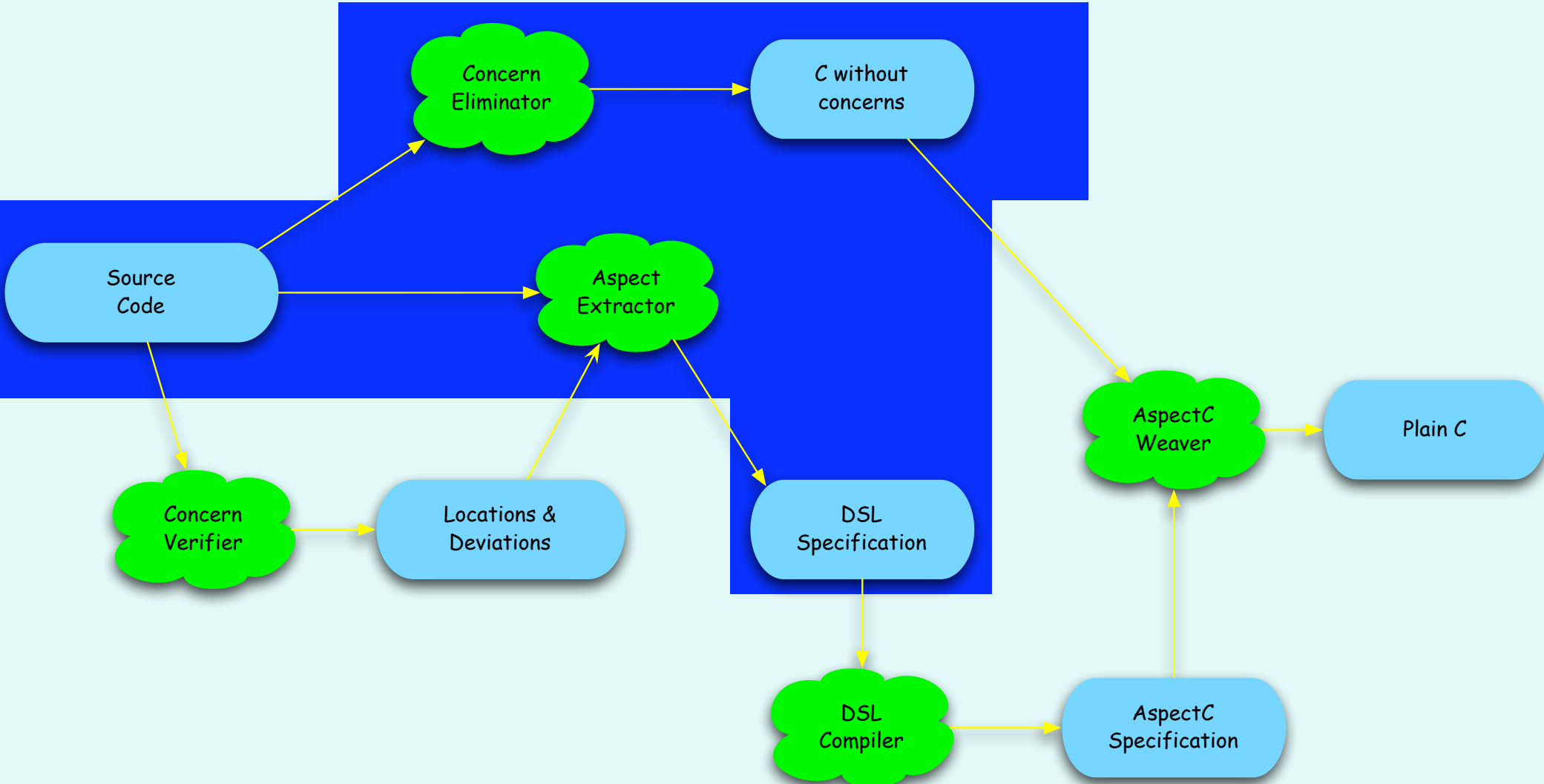


3rd Strategy

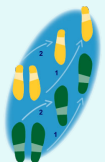
Automated weaving for all code



Automated weaving for all code

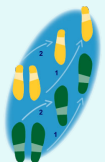


- ☑ Introduction to AOSD
- ☑ Goals of Ideals project & experiment
- ☑ Idioms-based approach
- ☑ Different adoption strategies for migrating toward AOSD approach
- ☑ Observations
- ☑ Conclusions

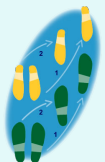


When applying the approach on the CC1 component

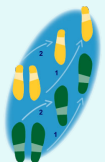
- ☑ code size reduced by 7%
- ☑ code size reduced by 84% for concern
 - ☑ 961 lines of C-code vs. 130 lines of DSL-code
- ☑ removed scattering
- ☑ removed code duplication
- ☑ Improved consistency & correctness
- ☑ DSL specification serves as documentation



- ☑ Consistency of DSL/C code
 - ☑ can be alleviated by only annotating deviations
 - ☑ provide annotations in structured comments
- ☑ Full approach is semi automatic
 - ☑ Concern verifier requires a domain expert
 - ☑ Some tools (Concern Eliminator, DSL Compiler) are not yet implemented



- ☑ Introduction to AOSD
- ☑ Goals of Ideals project & experiment
- ☑ Idioms-based approach
- ☑ Different adoption strategies for migrating toward AOSD approach
- ☑ Observations
- ☑ Conclusions



- ❑ Automatically migration idioms-based approach into AOSD approach is feasible
 - ❑ at least for homogeneous, non-tangled concerns
 - ❑ can be done gradually
- ❑ AOSD approach offers clear benefits in terms of quality attributes such as code size, correctness, consistency, etc ...

