Initial Results for Quantifying AOP

an technical report

Pascal Durr Lodewijk Bergmans Mehmet Aksit
University of Twente
{durr,bergmans,aksit}@ewi.utwente.nl

Abstract

This technical report reports on the initial result which were gathered from an controlled experiment, conducted on the the third of June, at ASML, in the Netherlands. This basically summarizes all information without much context information and much interpretation of the data.

1. Introduction

In [3] we reported on a case-study which we conducted at ASML. The goal of this case-study was to convince ASML that the usage of Aspect Oriented Programming would address crosscutting concerns. These crosscutting concerns were identified in an earlier phase of the Ideal project. Example of such crosscutting concerns are: Tracing, Error Handling and Contract Enforcement. The case-study not only resulted in an prototype implementation of an aspect-oriented weaver but also addressed several quality concerns which were expressed. Examples of these concerns are: compile-time performance, run-time performance, and the ability to use conventional debugging tools. As a direct result of the case-study a transfer project was initiated which the sole purpose to mature the tooling for industrial-strength usage.

In order to quantify the benefits of using AOP we conducted a controlled experiment, with 17 ASML software developers. This technical report present the initial results of the controlled experiment. It first discusses the details of the used crosscutting concern, in this case Tracing. Section 3 presents the experiment setup and the validation of equivalence of several details of the experiment, e.g. groep and scenarios. Finally, section 4 presents the initial results fo the experiment. This paper does nor (yet) draw any definitive conclusions about whether AOP is useful for ASML and especially in general.

2. Tracing Aspect

For this experiment we use the typical *Tracing* aspect. This aspect was the first driver to start developing *WeaveC*. Although this is at first seems to be a "simple" aspect, there are some detail which make them non trivial.

In [1] and [2], Bruntink et. al. already discussed some of the details of tracing, with respect to migration. WeaveC will first be used on newly developed code. As such an optimal tracing definition has been created.

For each function, the parameters should be traced. There is a subset of functions which cannot be traced, as the tracing framework has not been initialized at that time. Parameters refer to all (global)variables in the scope of this function, function arguments and return value of a function. We discuss, here two types; input and output parameters. Input parameters, are those parameters which are only read. Similarly, output parameters, are those parameters which are written. Some parameters can of course be both input and output parameters. Parameters which are passed to a function or where manipulation occurs via pointer operations, are considered to be written and thus output paramters. Tracing can thus be split up into two actions.

- Trace the function name and all input parameters at the start of a function.
- Trace the function name and all output parameters (including the return value) at the end of a function.

It should be noted that the inclusion of global variables is not in the previously accepted definition of tracing, which is currently being used within ASML. Therefor, any error with respect to the global variables, will be discarded.

WeaveC also supports annotations to allow the developers to deviate from the idiom. E.g. in case of performance issues. For aspect *Tracing* the following annotations are defined:

- $\$trace(TRUE \mid FALSE)$: this annotation is used to control if a module(file), function, parameter, variable or type should be traced.
- $\$trace\_as(fmt = \text{“}\ldots\text{”}, expr = \text{“}\ldots\text{”})$: this annotation is used to trace a parameter, variable or type in a different manner.

Aspect *Tracing* is defined as follows:

- All functions should be traced
  - Except for a fixed set of functions,
  - Except for those functions, which are annotated with $\$trace(FALSE)$ or whose module is annotated with $\$trace(FALSE)$.
- For each traceable function trace the input parameters at the start and the output parameters at the end of the function.
  - Except for those parameters, which are annotated with $\$trace(FALSE)$ or whose type is annotated with $\$trace(FALSE)$.

3. Experiment Setup

The experiment was included in a training which was given to a set of developers. The experiment consisted of two sessions of each half an hour. The first session was without *WeaveC* and the second session was with *WeaveC*. The subjects had to executed 5 simple change scenarios each session. Figure 1 present an overview of the training and the experiment.

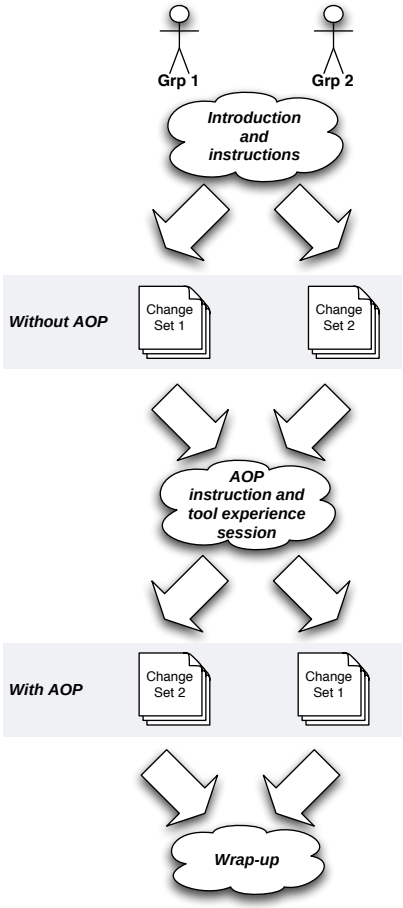

Figure 1. Overview of the experiment and training

3.1 Subjects

We split up the total group into two groups. The lack of facilites was one reason for this split up. The most important reason was to verify that the two change scenario sets were equivalent. The training was scheduled for one day with a morning and an afternoon session. In total there two twenty subjects.

We were also interested in finding some correlations between the results and the characteristics of the developers. Therefor, we have gathered the following characteristics:
Years of experience in software development : numeric
Years of experience at ASML : numeric
Gender : Male/Female
Age : numeric
Education Level : (4=PhD, 3= MSc, 2=BSc, 1=MBO)
C Level : (5=Expert,1=None)
Regular THXA user : Yes/No
Note that *MBO* is a more practical oriented education and *THXA* is the current tracing framework.

For the experiment we have the following subjects: It should me noted that two subject in the morning session were accidentally included in group one. This is why, in the morning group one has more subjects than group two.

3.1.1 Correlations

We found the following correlations between the subjects.

- There was a strong, negative correlation between the level of education and the experience with C [r=-.62, n=17, p$\leq$.05].
- There was a strong, positive correlation between the years at ASML and the regular THXA user [r=-.64, n=17, p$\leq$.05].
- There was a medium, positive correlation between the experience with C and the regular THXA user [r=-..49, n=17, p$\leq$.05].
- There was a strong, positive correlation between the age and the years in software development [r=-.85, n=17, p$\leq$.05].

3.1.2 Group Equivalence

On the bases of the previous properties we verify that the groups are equivalent in the characteristics.

Group	Characteristic	N	Min	Max	Mean	Std. Dev
1	Leeftijd	12	26	45	35.42	5.664
	Opleiding	12	2	3	2.50	.522
	JarenSW	11	1.0	20.0	9.000	5.7966
	JarenASML	12	.0	7.0	4.208	2.7091
	Clevel	12	1	5	3.50	1.087
	THXA	12	0	1	.58	.515
2	Leeftijd	5	26	43	33.60	6.348
	Opleiding	5	2	3	2.20	.447
	JarenSW	5	1.0	17.0	9.600	5.8566
	JarenASML	5	.0	5.0	1.460	2.0268
	Clevel	5	4	5	4.20	.447
	THXA	5	0	1	.60	.548

Table 2. Groups

From this we can observe the following:
- Subjects in group one are, one average, working longer at ASML.
- Subjects in group two are, one average more experienced in C.

3.1.3 Morning Afternoon Equivalence

On the bases of the previous properties we very that the groups are equivalent in the characteristics.

Time	Characteristic	N	Min	Max	Mean	Std. Dev
M	Leeftijd	10	31	43	36.80	4.131
	Opleiding	10	2	3	2.50	.527
	JarenSW	9	5.0	17.0	10.333	4.4721
	JarenASML	10	.0	7.0	4.380	2.8770
	Clevel	10	3	5	3.90	.738
	THXA	10	0	1	.60	.516
A	Leeftijd	7	26	45	32.14	6.866
	Opleiding	7	2	3	2.29	.488
	JarenSW	7	1.0	20.0	7.714	6.9213
	JarenASML	7	.0	5.0	2.000	2.0817
	Clevel	7	1	5	3.43	1.272
	THXA	7	0	1	.57	.535

Table 3. Time of day

From this we can observe the following:
- Subjects in the morning are, one average, working longer at ASML.
- Subjects in the morning are, one average, working longer in Software Engineering.

3.1.4 Environment and Tooling

The experiment and training was conducted in an external location. However, the subjects were able to login in remote to ASML, and conduct the experiments in there own build environment and on the build servers of ASML. Thus reducing any possible side effects of using another environment. Similarly, the build times of the system would be the real life delay, which they would also experience in there regular environment.

id	Morning/Afternoon	Group	Gender	Age	Edu.	Year SW	Year ASML	C level	THXA
mmM1	O	1	M	35	3	8	7	3	1
psM1	O	1	M	41	3	5	6	3	0
djM1	O	1	M	37	2	12	5.5	4	1
ydM2	O	2	M	29	3	5	0.5	3	0
vpM2	O	2	M	31	2	7.5	0.25	4	0
mfO2	O	2	M	43	3	17	0	4	0
rlM1	O	1	M	39	2	?	1	4	0
dhM1	O	1	M	32	3	6	6	3	1
dbM2	O	2	M	27	2	3	0.2	3.5	0
hpM1	O	1	M	39	2	15	7	5	1
csM1	O	1	M	39	2	15	4	5	1
jwM1	O	1	M	32	3	7	7	4	1
mvA2	M	2	M	32	2	10	1	5	1
mbA2	M	2	M	27	3	0	3	3	0
rgA2	M	2	M	36	2	12	5	4	1
mbA1	M	1	M	27	3	1	1	1	0
juA1	M	1	M	33	2	7	5	4	1
mkA2	M	2	M	26	2	1	1	4	1
caA1	M	1	M	26	3	3	1	3	0
wrA1	M	1	M	45	2	20	0	3	0

Table 1. Subjects

As explained earlier we used the WeaveC tooling, developed by ASML. The usage of this tooling has a performance penalty. We could have removed the tooling from the experiment. However, the tooling will provide feedback for syntax errors and such. This is vital for determining the kind of errors the subject made. We included the build fase into the experiment. To verify that the performance hit did not greatly influence the experiment, we also collected the build times, with and without WeaveC. The results are shown here:

#C builds	Average C build time	#AOP builds	Average AOP build time	Δ
65	24.71	88	51.82	210%

Table 4. Build times

We have decided to include the build times, although the build times are twice as large. This is apparently the overhead of using WeaveC while developing your source code.

3.2 Treatments

We use change scenarios as treatments to determine the possible gain of using AOP. ASML uses change request and problem reports for maintaining the software. We have chosen to use a canonical set of change scenarios. Each change request and problem report can be modelled as a composition of these scenarios. Due to the time limitations of the experiment we could only select five scenarios.

These change scenarios only apply if the resulting changes affect the idiom code, i.e. thus changing the core code without any affect on the idiom is not considered a change scenario.

We created the following list of scenarios:

- Add a function,
- Remove a function,
- Change a function - Add a parameter,
- Change a function - Remove a parameter,
- Change a function - Change a parameter,
- Add tracing to a function,
- Remove tracing from a function,
- Change tracing of a function - Do not trace a function,
- Change tracing of a function - Only trace a specific parameter,
- Change tracing of a function - Trace all parameters except one,
- Change tracing of a function - Also trace a global variable,
- Change tracing of a function - Trace a parameter in a different manner.

From this list we selected the following scenarios. This selection was made with input from ASML to decide which scenarios were the most occurring and valuable, according to ASML.

1. Adding tracing to a function,
2. Change a function - Change a parameter,
3. Remove tracing from a function,
4. Change tracing of a function - Only trace a specific parameter,
5. Remove a function.

For each scenario we will now show what is required to fullfil the scenario.

1. **Add tracing to a function**: This is similar to scenario 1.
   Old : Add tracing code to it, need to check for variable usage and null pointers.
   New : -
2. **Change a function - Change a parameter**: E.g. from int a to int* a, this requires adding a null pointer check.
   Old : Alter the tracing code, need to check for local/global variable usage and null pointers.
   New : -
3. **Remove tracing from a function**: This scenario is usually used to gain performance of the machine.
   Old : Remove trace statements from the a function or file
   New : Add no trace annotation
4. **Change tracing of a function - Only trace a specific parameter**:
   Old : Alter the trace statements to reflect this situation.
   New : Add no trace annotations to all parameters except the one which should be traced.
5. **Remove a function**: No benefits expected here
   Old : Remove function body
   New : Remove function body

3.2.1 Example scenario

We now show part of an instruction sheet for one scenario, in this case CS1_1.

1. Navigate to directory: LO/LOQM/int/bin:

```
cd LO/LOQM/int/bin
```

2. Check out file: LOEWdata.c:

```
cleartool co -nc LOEWdata.c
```

3. Locate the following function in this file: LOEWDA_set_wafer_state,
4. Generate tracing code to this function.

5. Build the file:

```
ccmake LOEWdata.osparc
```

6. In case of build errors, please go to step fix these and rebuild the system, until there are no more build errors.
7. Check in file: LOEWdata.c

```
cleartool ci -nc LOEWdata.c
```

Explanation:

1. First we navigate to the directory where this file is located. This file is in a local version tree for each subject.
2. Next we checkout the file which we possibly need to modify. We use wrappers to ClearCase tooling for this.
3. In this file we have to locate the function which we possible need to modify.
4. In this scenario we have to generate tracing for this function.
5. Next we build this file using the build environment and build servers at ASML.
6. Repeat steps 4 and 5 until satisfied and successful compilation.
7. Checkin the modified file. In case of no changes, we asked the subjects to add some whitespace to the file.

3.3 Objects

The code we are using for our experiments, is actual code taken from the ASML codebase. We have explicitly chosen to take a different component as the component most of the subjects are currently working on. This prevents a learning affect and thus increases the significance of the experiment. As the idioms we are targeting are system wide idioms, the subjects should not feel alienated with the code. As described in the setup, we have two code samples which are used in the experiment. We have to verify that these sets are indeed equivalent, complexity wise. We used two metrics to verify that the code samples are not only equivalent bus also representative ASML code. We used three metrics for represented the perceived complexity of the code. These were the McCabe cyclometric complexity metric, the Halstead metric and a metric representing the number of arguments. We introduced the latter as tracing directly relates to the number of arguments. Due to confidentially reasons we cannot expose the metrics. We did however verify that code samples were indeed equivalent and representative for the ASML codebase.

3.4 Measurements

The goal of the experiment is to determine whether using AOP speeds up the development and maintenance of the ASML codebase. In order to quantify this we measure the following elements for each change scenario, with and without AOP.

- Time it takes to implement each change scenario
- Errors in the implementation of the change scenarios

3.5 Hypotheses

$H_0 : time_{old} \leq time_{new} \wedge errors_{old} \leq errors_{new}$
$H_1 : time_{old} > time_{new} \wedge errors_{old} > errors_{new}$

3.6 Variables

3.6.1 Independent Variable

- Years of experience in software development
- Years of experience at ASML
- Gender
- Age
- Education

3.6.2 Dependent Variables

- Time to execute a change scenario
- Errors in the result of a change scenario

4. Experiment Results

Before we present the results in details we show which persons or change scenarios were excluded from before continuing further statistical analysis.

4.1 Outliers - Persons

From our original 20 people, we have discarded 3 three of them.

dmM2 : The data from this subject lacked accurate checkin and checkout time stamps.

ydM2 : The checkin and checkout time stamps for the second session were missing.

mbA2 : This subject was a member of the Ideals research project which was there solely for training purposes.

4.2 Outliers - Scenarios

Once the three persons were removed from the data set and we continued to check for any outliers per change scenario. For each change scenario we discuss why we excluded this from the set. We excluded people based on two criteria, whether they were outliers from the statistical analysis, or whether the time it took them exceeded 1200 seconds.

Once we removed the outliers from a time perspective we had no further outliers from the error perspective.

4.2.1 Change Scenario 1 - Session 1

3 - hpM1 : extreme statistical outlier
9 - juA1 : exceeded 1200 seconds mark
11 - caA1 : exceeded 1200 seconds mark
12 - wtA1 : extreme statistical outlier

4.2.2 Change Scenario 2 - Session 1

4 - jwM1 : statistical outlier

4.2.3 Change Scenario 3 - Session 1

None

4.2.4 Change Scenario 4 - Session 1

1 - dhM1 : statistical outlier

4.2.5 Change Scenario 5 - Session 1

4 - jwM1 : extreme statistical outlier

4.2.6 Change Scenario 1 - Session 2

10 - mbA1 : statistical outlier
14 - vpO2 : extreme statistical outlier

4.2.7 Change Scenario 2 - Session 2

None

4.2.8 Change Scenario 3 - Session 2

9 - juA1 : statistical outlier

4.2.9 Change Scenario 4 - Session 2

None

4.2.10 Change Scenario 5 - Session 2

10 - mbA1 : statistical outlier
2 - djM1 : statistical outlier

4.3 Initial processing

As the number of subjects is really low and it is hard to get a statistical significant value. We also show the absolute values taken from the experiments.

Pair	#CS1	#CS2	CS1	CS2	Average CS1	Average CS2
CS1_1 - CS2_1	12	15	6416	3877	534.67	258.47
CS1_2 - CS2_2	12	15	5914	6323	492.83	421.53
CS1_3 - CS2_3	9	11	1290	2740	143.33	249.09
CS1_4 - CS2_4	8	13	2332	2948	291.50	226.77
CS1_5 - CS2_5	17	17	4250	3137	250.00	184.53

Table 5. Times

Pair	#CS1	#CS2	CS1	CS2	Average CS1	Average CS2
CS1_1 - CS2_1	12	15	9	0	0.75	0.00
CS1_2 - CS2_2	12	15	27	0	2.25	0.00
CS1_3 - CS2_3	9	11	0	0	0.00	0.00
CS1_4 - CS2_4	8	13	11	12	1.38	0.92
CS1_5 - CS2_5	17	17	0	0	0.00	0.00

Table 6. Errors

It should be noted that in the manual version there were 7 critical errors, which could have resulted in a segmentation fault during runtime, were made in the without AOP session.

4.4 Development Time

4.4.1 Descriptives

CS	N	Min	Max	Mean	Std. Dev.
CS1_1	11	254	902	583.27	227.267
CS1_2	12	76	648	412.42	176.887
CS1_3	9	91	210	143.33	44.766
CS1_4	7	142	339	249.29	71.979
CS1_5	16	84	540	211.19	117.175
CS2_1	15	106	553	258.47	129.040
CS2_2	15	153	910	421.53	243.497
CS2_3	10	59	333	216.60	85.822
CS2_4	13	89	421	226.77	119.703
CS2_5	15	106	253	179.40	46.417

Table 7. Times

4.4.2 Significance

We calculated the significance through the use of a standard paired sample test with a confidence interval of 95%.

Pair	N	Mean	Std. Dev.	Sig.
CS1_1 - CS2_1	9	414.11	306.28	.004
CS1_2 - CS2_2	11	-46.64	369.54	.684
CS1_3 - CS2_3	6	-91.33	-91.34	.031
CS1_4 - CS2_4	6	94.67	78.26	.031
CS1_5 - CS2_5	14	43.07	119.64	.201

Table 8. Significance of Times

Observations:

- Adding tracing to a function takes considerably less time with AOP than without.
- Removing tracing from a function takes more time with AOP than without. This is probably caused by the (first) usage of annotations.
- Selective tracing parameters in a function, takes less time with AOP than without.

4.4.3 Correlations

We found the following correlations between the subjects and the difference between without and with AOP.

- There was a strong, positive correlation between the difference in time of scenario 2 and the number of years in software development [r=-.624, n=11, p$\leq$.05].

And per scenario:

- There was a strong, positive correlation between the time of scenario 1 without AOP and the number of years at ASML[r=.702, n=11, p$\leq$.05].
- There was a negative correlation between the time of scenario 5 without AOP and the number of years at ASML [r=-.511, n=16, p$\leq$.05].
- There was a strong, positive correlation between the time of scenario 1 without AOP and the time of scenario 3 without AOP [r=.704, n=11, p$\leq$.05].
- There was a strong, positive correlation between the time of scenario 1 and the number of years at ASML[r=-.906, n=9, p$\leq$.05].
- There was a strong, negative correlation between the time of scenario 2 with AOP and the number of years of software development [r=-.638, n=14, p$\leq$.05].
- There was a strong, negative correlation between the time of scenario 2 with AOP and age of the subjects [r=-.648, n=14, p$\leq$.05].
- There was a strong, negative correlation between the time of scenario 3 with AOP the number of years at ASML [r=-.636, n=10, p$\leq$.05].
- There was a strong, negative correlation between the time of scenario 3 and the usage of the THXA tracing framework [r=-.751, n=10, p$\leq$.05].

4.5 Errors

4.5.1 Error classification

We made the following error classification:
0 : No errors,
1 : Typo in tracing text string,
2 : Less tracing,
3 : Wrong tracing,
4 : No parameter checking code.

4.5.2 Descriptives

CS	N	Min	Max	Mean	Std. Dev.
CS1_1	11	0	3	.82	1.250
CS1_2	11	0	4	2.45	1.809
CS1_3	9	0	0	.00	.000
CS1_4	7	0	4	1.57	1.618
CS1_5	16	0	0	.00	.000
CS2_1	17	0	0	.00	.000
CS2_2	15	0	0	.00	.000
CS2_3	10	0	0	.00	.000
CS2_4	13	0	3	.92	1.441
CS2_5	15	0	0	.00	.000

Table 9. Errors

4.5.3 Significance

We calculated the significance through the use of a standard paired sample test with a confidence interval of 95%.

Observations:

- Adding tracing to a function manually introduces significantly more errors than with AOP.
- Changing the signature of a function manually introduces significantly more errors than with AOP.

Pair	N	Mean	Std. Dev.	Sig.
CS1_1 - CS2_1	11	.818	1.250	.055
CS1_2 - CS2_2	10	2.300	1.829	.003
CS1_3 - CS2_3	6	0	0	-
CS1_4 - CS2_4	6	.500	2.588	.656
CS1_5 - CS2_5	14	0	0	-

Table 10. Significance of Errors

4.5.4 Correlations

We found the following correlations between the subjects and the difference between without and with AOP.

- There was a strong, positive correlation between the difference in errors of scenario 1 and the number of years in software development [r=-.694, n=10, p$\leq$.05].
- There was a strong, positive correlation between the difference in errors of scenario 3 and the number of years in software development [r=-.882, n=6, p$\leq$.05].
- There was a strong, positive correlation between the difference in errors of scenario 3 and the age of the subjects [r=-.866, n=6, p$\leq$.05].

And per scenario:

- There was a strong, negative correlation between the erros of scenario 1 without AOP and the number of years of software development [r=-.716, n=11, p$\leq$.05].
- There was a strong, negative correlation between the erros of scenario 2 without AOP and the education level [r=-.635, n=11, p$\leq$.05].
- There was a strong, positive correlation between the erros of scenario 2 without AOP and the years of software development [r=-.643, n=11, p$\leq$.05].
- There was a strong, negative correlation between the erros of scenario 4 without AOP and the number of years at ASML [r=-.726, n=13, p$\leq$.05].
- There was a strong, positive correlation between the erros of scenario 4 without AOP and the usage of the THXA framework [r=-.639, n=13, p$\leq$.05].

Acknowledgments

This work has been partially carried out as part of the Ideals project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Senter program. This work is supported by European Commission grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe).

References

[1] M. Bruntink, A. v. Deursen, M. D'Hondt, and T. Tourwé. Simple crosscutting concerns are not so simple – analysing variability in large-scale idioms-based implementations. In *Proceedings of the Sixth International Conference on Aspect-Oriented Software Development (AOSD'07)*. ACM Press, March 2007. To appear.

[2] M. Bruntink, A. v. Deursen, and T. Tourwé. Isolating idiomatic crosscutting concerns. In *Proceedings of the International Conference on Software Maintenance (ICSM'05)*. IEEE Computer Society, 2005.

[3] P. Durr, G. Gulesir, L. Bergmans, M. Aksit, and R. van Engelen. Applying aop in an industrial context. In *Workshop on Best Practices in Applying Aspect-Oriented Software Development*, Mar 2006.