SMA:
Software Modeling and Architecture

Q&A Session

Categories

8 questions in “**GT / Pharo / Smalltalk**”

1 question in “**Reflection**”

7 questions in “**Static & Dynamic Analysis**”

2 questions in “**Software Metrics**”

1 question in “**Visualizations**”

2 questions in “**Code and Test Smells**”

8 questions in “**Fuzzing**”

1 questions in “**Organizational Affairs**”

In total 30 questions in 8 categories

GT / Pharo / Smalltalk

GT / Pharo / Smalltalk – Q01

Q: What is the difference between |x| and x := nil in Pharo and why does the first seem to be commonly done, rather than directly instantiating the variable (i. e. x := 1).

A:

1) These are different things. |x| is the variable declaration like

```
     int number;
```

in Java or C. It is required, but only when you write code within methods. The playground does the declaration for you in the background in most cases. That’s why you don’t find many playground snippets with those declarations. On the other hand, those declarations are very prevalent in method implementations.

2) nil assignments (`x := nil`) are a good practice to emphasize that a variable has not yet been initialized. Same for Java, e.g.,

```
Integer currentValue = null;
```

GT / Pharo / Smalltalk – Q02

Q: What kind of Smalltalk code do we need to know and what do we not need to know by heart. For example, do we need to know the rather long code to make a Roassal visualisation, which has been used in the exercises? How about Live Document code, which is not very complex but we have hardly used?

A: You do need to know the basic structure and concepts of Smalltalk code. For example, you should know how to use blocks ([…]) , conditions (ifTrue: …), iterate over collections (collect, do, …), how to use `compile: aString`. You also should be able to understand the code snippets in the assignments.

Moreover, you should be familiar with GT. For example, you should know what the purpose of a pragma or a live document is and how to use it.

You do <u>not</u> need to know in detail the APIs of libraries such as Roassal or Moose.

GT / Pharo / Smalltalk – Q03

Q: What are advantages of Smalltalk/Pharo/GT when analyzing a software system versus other tools/languages?

A:

It has a very nice plotting features and can create vectorized graphs. This is awesome for publications.

It is very interactive and the tool can also be used by stakeholders. For example, a stakeholder can click in the plot to see more details for a specific pie chart section.

GT / Pharo / Smalltalk – Q04

Q: Why is it hard to calculate CYCLO for Smalltalk methods?

A: To my knowledge, it has not been added to the base system. So you would need to add the logic to the `Class` class.

The problem comes from the fact, that instances might have different methods compared to their classes. Even more, traits and extension methods inject code defined in other places. Those features could be used in many classes and introduce additional complexity in each class that uses them, although the implementation resides only in one place.

So the outcome would be heavily biased without complex preprocessing.

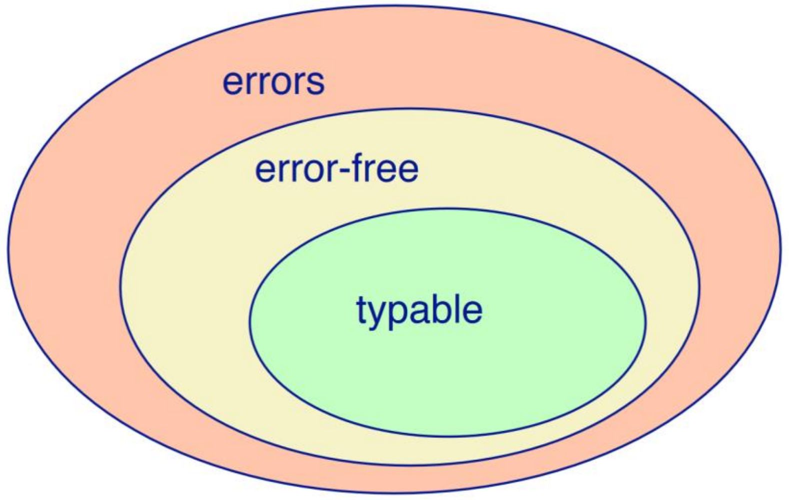
NB: What we had is the cyclomatic complexity for Moose models. Moose has support for cyclo built-in.

GT / Pharo / Smalltalk – Q05

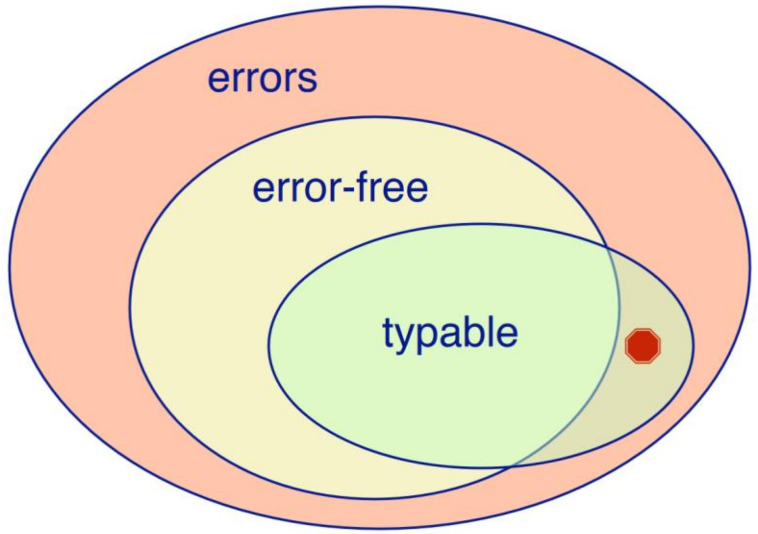
Q: Can a type system reject only programs that lead to type errors?

A: A typed system is these days intrinsic (embedded into the language) and does not necessarily detect other types of errors (imagine I/O related errors). Moreover, sometimes the type system is unsafe and it does not detect all the type errors.

See example with Java:



Type safety is sound



Type safety is **un**sound

An unsound type system can accept code that may lead to run-time errors.

Example:
co-variant arrays in Java

https://dzone.com/articles/covariance-and-contravariance

GT / Pharo / Smalltalk – Q06

Q: 1) Why are Behavior, ClassDescription and Class implemented in separate classes?

2) Why does Metaclass not need the functionality of class? Is it because a Metaclass has only one single instance?

A:

1) It was a design decision to separate the concerns. As a result, it improves code reuse. This leads to better modularity, portability and maintainability.

2) No, it is due to the separation of concerns. A metaclass is a controller and does not care that much about the corresponding class logic.

Each metaclass shares the class variables of its instances. Hence, it is mandatory to only have one instance that has control over all the shared variables for a specific class.

GT / Pharo / Smalltalk – Q07

Q: What is an example of a non-meta-circular language?

A: The ability to support meta-circularity requires extremely high abstractions (you need objects that can represent every piece of the language and the environment).

Hence, you can argue that rather primitive languages without any abstractions do not have any meta-circular abilities.

A typical example would be assembly.

If you ask for a “not fully meta-circular language” you can use any language that has reflection abilities, but does not support access to all the features offered by the language itself. For example, in Java you can access methods with reflection, but you cannot modify variables without changing the environment.

More details: http://www.jot.fm/issues/issue_2020_03/article11.pdf

GT / Pharo / Smalltalk – Q08

Q: How would you get all possible pragmas in GT?

What is exactly a pragma? Can it be compared with an annotation telling some clients that it is capable of doing certain things (e.g. search itself or display itself)?

A: You could use the `PragmaCollector` to get all pragmas in GT. It essentially collects all methods that contain a certain pragma. Every time you add a pragma it gets notified and automatically adds/removes your pragma to/from its list.

A pragma is a text string enclosed with angle brackets, for example:

`<samplePragma>`

The actual pragma itself is represented as class instance. It basically resembles the concept of annotations in Java.

Reflection

Reflection – Q09

Q: Lecture 4, Slide 6 & 7

We defined on slide 6 Introspection as structural reflection and Intercession as behavioral reflection. However on Slide 7, it is stated structural reflection could only reason about abstract data types. Why?

I mean Introspection is the ability of a program to observe and reason about it's own state. This means you can get during runtime the type of a variable, it's concrete data type. What do I understand wrong?

A: The term "abstract data type" has been used from a language implementation point of view.

abstract = something not based on any particular thing

$\Downarrow$

= hardware implementation in CPUs

⇨ abstract data type = data type offered by the programming language (that does not necessarily exist in hardware)
more info: https://en.wikipedia.org/wiki/Abstract_data_type

Static & Dynamic Analysis

Static & Dynamic Analysis – Q10

Q: SMA-09-StaticAnalysis, Slide 10:

Here it states that dynamic analysis only leads to false negatives.

But can’t we just turn around the question to get false positives?

For example, if we want to check whether some parts of the code are live. Instead of asking the question “is the code x live?”, we could ask “is the code x dead?”. With this, we would get a false positive in dynamic code analysis. (In dynamic analysis, it may happen that we don’t execute all parts and classify them as dead. This would then be a false positive (classified as dead, but it is in fact live)).

A: You are technically right about the inversion, but it is not the whole story:

The idea was to illustrate that you can only analyse what you executed in a limited number of test runs. Hence, you will most certainly encounter missed code paths in more complex applications (= false negatives, you overlooked something).

However, how these false negatives end up with your criteria (“is code x live”/”is code x dead”) is another story.

In the end, it really depends what you refer to as “dynamic analysis”. Is it only the analysis of the system with the extraction of values, or is it also their evaluation?

Static & Dynamic Analysis – Q11

Q: SMA-01-Intro, Slide 40:

In the first lecture, test coverage is mentioned as a task of dynamic software analysis. 1) Does this mean that tests are also a form of dynamic program analysis? 2) Or are they rather part of the software itself?

A:

1) When talking about (coverage) tests, people usually refer to JUnit tests with EclEmma (or similar). These tests execute code, hence they are a form of dynamic program analysis. Nevertheless, coverage analysis can be performed statically or dynamically. However, dynamic analysis is usually preferred due to much higher execution speed.

2) Such tests are exploiting dynamic analysis techniques and they are also part of the software/application (depending on what you consider to be an application).

application = a program designed to do a particular job; a piece of software

Static & Dynamic Analysis – Q12

Q: There is a set of slides labeled "SMA-09-StaticAnalysis-ML.pdf". I don't think we have looked at these during the lecture. Do we need to study these?

A: No.

Static & Dynamic Analysis – Q13

Q: When are class invariants executed (discovered/enforced?) in the development flow? When the tests are running? During monkey testing?

It was stated that checking the invariants is usually turned off in the Test and Production environment for performance reasons. Also it would not make sense as the invariants could alter the program execution (or even crash).

A: There are two possibilities: either you know that a value must be within certain bounds and you specify the invariants manually during development (e.g., ambient temperature sensor -30°C < x < 60°C), or you let software automatically discover invariants in your application during execution. The former is usually done by adding constraints to improve the code reliability, while the latter is rather done to find outliers/issues (you are not sure whether the detected invariant is correct, it is just an observation). The latter also supports monkey testing.

The invariants might be constantly verified in security-sensitive applications (software for money transactions, ...). This requires a special execution mode with additional privileges for the execution environment of the application, similar to the debug mode. Ordinary applications use this special mode only when under test, if at all.

Static & Dynamic Analysis – Q14

Q: Are there any specific guidelines when doing a Hybrid analysis (Static + dynamic)? Such as steps to take: which analysis first, how both analyses iterate between each other?

A: Not really; experience is the key.

In most cases you can design an analysis using any of the three, but usually the two other have major drawbacks (speed, resource usage, ...).

So you end up with one preferred analysis type for a specific task.

Example:

Imagine you want to find a method with the name "helloWorld()".

Static:	**iterate over all methods with the String "helloWorld()"**
Dynamic:	**collect all called methods with the name "helloWorld()"**
Hybrid:	**use the static technique for source code, and the dynamic technique for compiled code**

Which one to select? Static is comprehensive, dynamic is easy to maintain and efficient over short-term, hybrid is even more comprehensive than static. The choice depends on several external factors.

Static & Dynamic Analysis – Q15

Q: Does Intraprocedural and Interprocedural analysis differ in a OOPL vs a procedural PL? If so how?

Reason reading up on the topic I found the definition here (IBM, https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/ilec/bindopt.htm) of intraprocedural analysis and there the term depends on the definition of a compilation unit in the language.

A: Intraprocedural and interprocedural analyses differ in their scope. Intraprocedural analyses happen within a function, interprocedural across functions. Now one question remains: what is a function?

function =	a sequence of program instructions that performs a specific task, packaged as a unit.

Back to OOPL and PPL, they might have different rule sets (classes vs modules) and "units" (functions vs submodules), but the general idea remains the same.

Static & Dynamic Analysis – Q16

Q: 1) Is a class instantiation/ class method invocation of another class inside a class method intraprocedural? 2) What if it was a private class and thus in the same compilation unit as the observed class?

```
fnc x() {
        var y = new Class()
        if y.t() {
        } else {
        }
}
```

A: 1) Think abstract syntax tree (AST): in your favorite IDE, every code snippet is represented as AST. Iterating over all AST nodes that belong to a specific method is intraprocedural (all nodes are attached to the method declaration node). However, if you resolve an "external call node"/"instantiation node" you will just turn your analysis into an interprocedural one, because you rely on information from another function.

2) "private" does technically not change the function boundaries. However, there is an exception: private methods might become inlined by the compiler, therefore the function boundaries could change.

We won't go into more details here: this could be another entire lecture.

Software Metrics

Software Metrics – Q17

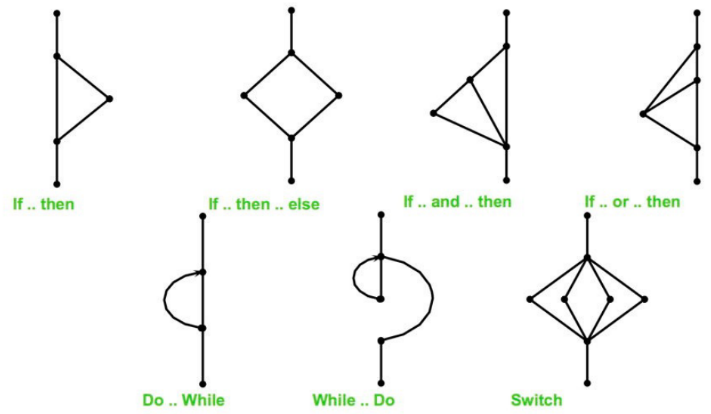
Q: SMA-07-MetricAndProblemDetection, Slide 7:

Why does a "do-while" have a different cyclomatic complexity than a "while-do"? Does this have a special reason (e.g. the program runs faster), or is it just a Definition? Furthermore, what would be the cyclomatic complexity of a for-loop?

A: Do-while, while-do, for-loop have the same cyclo value.

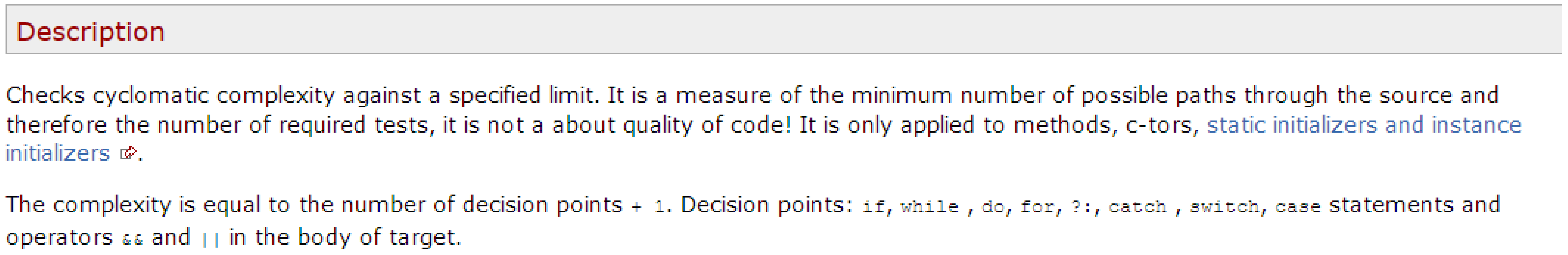
The number of independent linear paths through a program. (McCabe '77)

+ Measures minimum effort for testing



This might clarify:

https://checkstyle.sourceforge.io/config_metrics.html#CyclomaticComplexity



Description

Checks cyclomatic complexity against a specified limit. It is a measure of the minimum number of possible paths through the source and therefore the number of required tests, it is not a about quality of code! It is only applied to methods, c-tors, static initializers and instance initializers.

The complexity is equal to the number of decision points + `1`. Decision points: `if`, `while` , `do`, `for`, `?:`, `catch` , `switch`, `case` statements and operators `&&` and `||` in the body of target.

Software Metrics – Q18

Q: For cyclomatic complexity, what is and what is not a node?

A:

A node represents a statement that introduces a change in the control flow.

Examples:

conditions, loops, ...

What is not a node?

All remaining elements that do not change the control flow.

Examples:

a = a + 2;

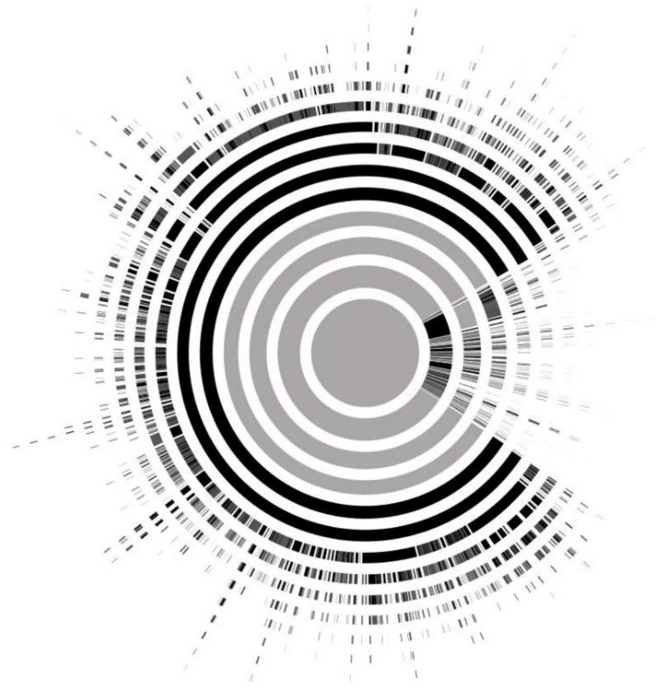
int double = 2;

...

Visualizations

Visualizations – Q19

Q: Exercise 3 – Visualisations - point a). This is one of the questions that I couldn't answer correctly during the mock exam. The explanation on the PDF it is still not clear for me. It is still hard for me to understand why this graphic is presenting this irregularity on one side.



A: As written in the mock exam exercise description, the center represents the Object class. The ring which is closest to Object represents classes which directly inherit from Object. We can see that some small classes are on the same hierarchy level (= in the same ring), but because they are black they contain tests. Consequently, those classes do not inherit from Class (ring 3) and are rather alien-like.

Code and Test Smells

Code and Test Smells – Q20

Q: The slides note that an eager test tests "more than one method". The solution to the exercise on eager tests however notes that "a single test should use a single assert statement". If a test uses multiple assert statements, but only on the same method (for example because that method increments a counter upon which the return is based), is this still considered an eager test? If yes, why is it considered a test smell to test the same method in one test multiple times?

A: In the literature I found only the statement "more than one method". However, I still argue that a single test should use a single assert statement (to be not eager).

IMHO: Your test example for the counter method should be implemented in a different way. Use setUp() and shutDown() methods to prepare the method under test. Then you will end up with only one assert statement in your test. As for every smell: sometimes it makes sense to just accept them as they are often a tradeoff between different properties.

Code and Test Smells – Q21

Q: Generally, are God classes good approach or should they always be separated into modules?

A: God class = class that knows too much = should know less.

Consequently, you should try to divide the class into different parts and then see what you can optimize from there.

Again: Sometimes it makes sense to just accept the smell as it is.

Example:

A property class that stores many variables (localization, configuration, ...) used in many different parts of the program might be acceptable.

Fuzzing

Fuzzing – Q22

Q: Is monkey testing also a type of fuzzing?

A: It depends. Fuzzing is an *automated* software testing technique.

When you do manual monkey testing it is technically not fuzzing, but when you use some kind of automation to perform the monkey testing I guess you could call that fuzzing.

Fuzzing – Q23

Q: Is fuzzing a type of dynamic program analysis?

A:

Dynamic program analysis = monitoring running application.

Fuzzing = creating input and monitoring running application.

Actually, it is more than that:

Fuzzing leverages dynamic program analysis among other techniques to find results.

Fuzzing – Q24

Q: Considering that one of the limitations of Symbolic execution is about the infinite execution tree that we can have, is it possible really use this execution in real programs or it is more a theoretical approach that it is really used only when we mix it with a concrete execution (Concolic Execution)

A: The infinite execution tree limitation is usually mitigated with the introduction of recursion depth threshold values. For example, only up to 10 recursions are allowed; after that recursions are skipped.

With such thresholds symbolic execution can be used in practice.

Fuzzing – Q25

Q: As I see it, fuzz testing tools are still experimental and are more of a nice add-on for security-critical systems rather than an everyday software devloper tool.

Do you agree? Are there arguments against this hypothesis?

A: They are not anymore experimental. You can find many different implementations from rather simple ones to even some with deep learning to create better input. https://arxiv.org/pdf/2010.12149.pdf

They are used these days mostly for penetration testing (and found numerous remarkable critical bugs), but not within development processes. If API/library developers would use such tools many security issues could be mitigated before they hit the market.

I agree that they particularly make sense in code where external input is processed (parsers of any kind, decoders, web APIs, ...). However, these days almost everything seems to work with such input.

Fuzzing – Q26

Q: In which stage of the project should we implement Fuzz testing? Also, what is the percentage of resources that would be reasonable for fuzz testing?

A: Fuzz testing requires a running application. So the testing should happen in your testing phase. It could be integrated into your testing pipeline.

There is no single answer regarding the resources required for fuzz testing. Fuzz testing could take several weeks depending on the configuration. The efficiency could be improved with white or grey box fuzzing that leverages information from the code to find better inputs.

Fuzzing – Q27

Q: How can we decide which fuzz testing to use?

A: If you don’t have access to any information of the target app you can only use black box fuzzing. If you have access to information, grey box fuzzing can improve the performance, and white box fuzzing can improve the fuzzing performance one more time.

In practice, white box fuzzing is usually unavailable (closed source apps) and grey box fuzzing is preferred over black box fuzzing due to the better performance.

Example:

You have an application that interprets HTML input (e.g., a web browser). So you know that the application will process HTML and you can specify the grey box fuzzer to generate HTML like inputs.

Fuzzing – Q28

Q: Why do we use hybrid fuzzing?

A: To improve the fuzzing performance.

Same story as with hybrid code analysis (compared to static and dynamic analysis).

Fuzzing – Q29

Q: What is the difference between white box black box and gray box (fuzz?) testing?

A: Fuzzing is the process of creating new or altered input and then monitoring apps given the generated input.

Black box fuzzing: fuzzer knows nothing about the target application and its input.

White box fuzzing: fuzzer leverages data from program analysis and constraint solving to improve the quality of the results.

Grey box fuzzing: fuzzer has some knowledge about the program, i.e., it is a combination of black and white box fuzzing.

See slide Fuzzing Q27 slide for an example with grey box testing.

Organizational Affairs

Organizational Affairs – Q30

Q: Do I have to send my written exam back to you by Swiss Post (snail mail)?

A: We are currently investigating this question, but did not yet receive any response. For now: please keep your exams until you got the final grade in your study report.

If we would need them, we will contact you by mail and/or Piazza.

Meta-information

Meta-information 01

Answer every aspect of a question!

Example 1:

Where would you like to go over holiday, and what is the last time you enjoyed holidays?

1) I would like to go to Hawaii. 2) It was several months ago.

Example 2:

Do you believe in “an apple a day keeps the doctor away”? Justify.

1) Yes, 2) because I always eat an apple a day and I never had a Corona infection.

Meta-information 02

Details about the exam:

Time:	**60 minutes**
Total points:	**60**
# exercises:	**6**
Exercise topics: (preliminary)	**1) General questions** **2) Smalltalk** **3) Smalltalk** **4) Random Lecture Topic** **5) Random Lecture Topic** **6) Random Lecture Topic**

The last lecture (Socio-technical Aspects in Software Systems) is not relevant for the exam.

… still more questions?

Next Time: Virtual Exam

60 minutes (+ set up) / CLOSED BOOK

16-Dec-2020, 10:15am until 12:15pm

(Zoom meeting link and last minute changes (if any) will be published in Piazza)

1. Join on time

2. Have your student ID ready

3. Don't forget a blue or black ball pen AND empty sheets of paper

4. Not allowed: pencils, internet, books, printouts, pocket calculators, mobiles, smart anything, ...

Thank you
VERY MUCH
for your
participation!

Good Luck!