ESE
exam preparation

Andrea Caracciolo
http://scg.unibe.ch/staff/caracciolo

Exam

- 7th January, 2016 — ExWi A6 @ 10.00
- **Register on KSL!**
- Exam: 60% of final grade
- Language
  - Q: English
  - A: English (preferred); German (possible)

Material

- It covers the material of the lectures (inc. guest lectures).

- simple knowledge questions + reasoning questions

- Suggested complementary material:

  - Sommerville, Software Engineering (7th-9th edition)

- You can **NOT** bring: books, slides, personal notes, electronic devices

Recommendation

- Answer questions at the end of each lecture slides

- Use the book to complement material presented during the lecture

Topics

- Terminology
- Software design/quality (principles & diagrams)
- Software Engineering Processes
- Software architecture (styles & properties)
- Testing (methods & techniques)

Topics

- **Terminology**
- Software design/quality (principles & diagrams)
- Software Engineering Processes
- Software architecture (styles & properties)
- Testing (methods & techniques)

Exercise

Terminology

1. define: architectural style
2. define: principle of encapsulation
3. Class vs. Object
4. Fault tolerance vs. Fault avoidance
5. define: Req. Consistency; Completeness; correctness

Exercise

Terminology

1. define: architectural style

An <u>architectural style</u> defines a family of systems in terms of a pattern of structural organization.
More specifically, an architectural style defines a ***vocabulary of components and connector types, and a set of constraints on how they can be combined****.*

Exercise

Terminology

1. define: architectural style

2. define: principle of encapsulation

Keep behavior together with any related information

Exercise

Terminology

1. define: architectural style

2. define: principle of encapsulation

3. Class vs. Object

***class**: a class is a template to create objects, it defines the state and methods of its instances, mainly exists at compile time*

***object**: has state and operations, an object is an instance of a class, it exists at runtime*

Exercise

Terminology

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of some of its components.
Fault avoidance seeks to prevent faults from being introduced into the software.

4. Fault tolerance vs. Fault avoidance

5. define: Req. Consistency; Completeness; correctness

Exercise

Terminology

Consistency: Are there any *requirements conflicts*?
Completness: Are *all functions* required by the customer included?
Correctness: Are the requirements correct?

5. define: Req. Consistency; Completeness; correctness

Topics

- Terminology
- **Software design/quality** (principles & diagrams)
- Software Engineering Processes
- Software architecture (styles & properties)
- Testing (methods & techniques)

Software design/quality

You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

- A user can book multiple one-way trips.
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight. Flights takeoff and land in an airport and occur on a specific date and time.

Exercise

Draw a UML class diagram describing the main domain abstractions outlined in the above specification.

Software design/quality

You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

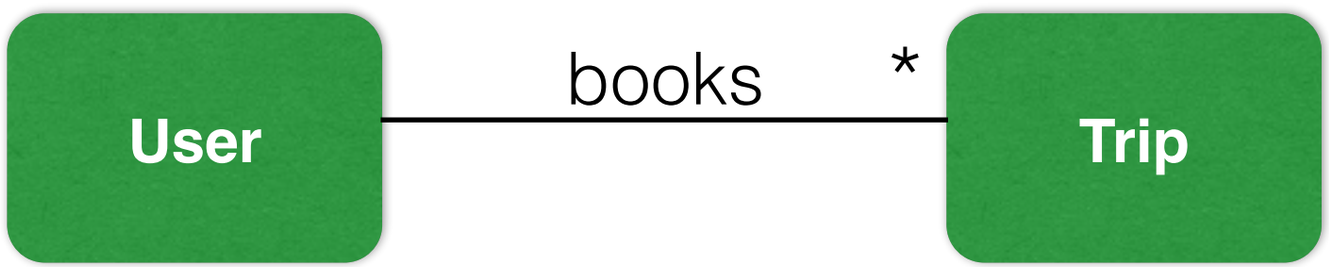
- A user can book multiple one-way trips
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight. Flights takeoff and land in an airport and occur on a specific date and time.

find classes

You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

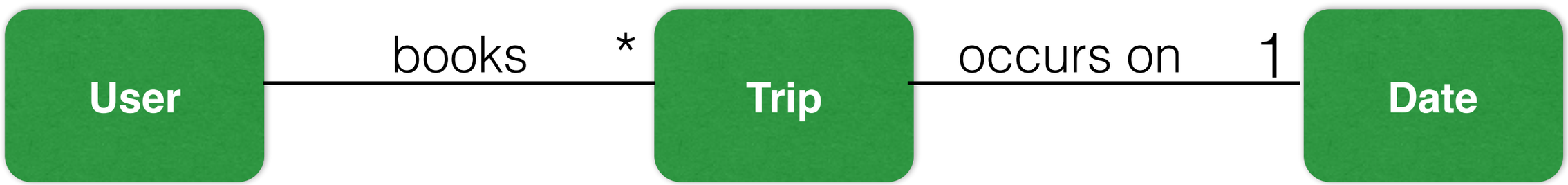
- A user can book multiple one-way trips
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight. Flights takeoff and land in an airport and occur on a specific date and time.



You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

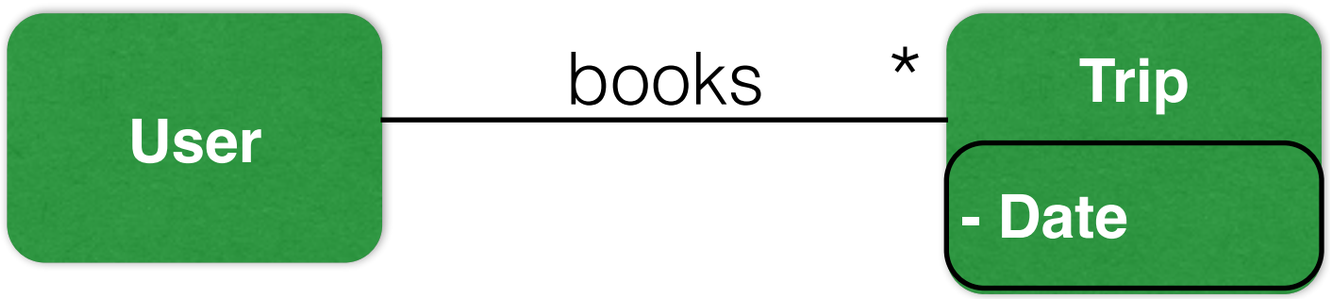
- A user can book multiple one-way trips
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight. Flights takeoff and land in an airport and occur on a specific date and time.



You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

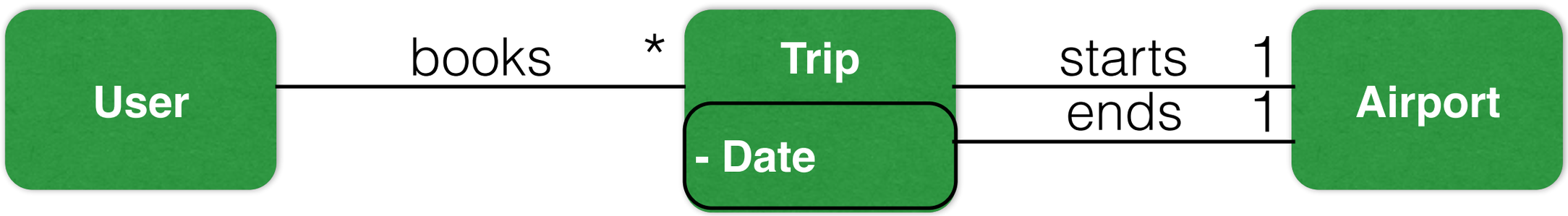
- A user can book multiple one-way trips
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight. Flights takeoff and land in an airport and occur on a specific date and time.



You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

- A user can book multiple one-way trips
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight. Flights takeoff and land in an airport and occur on a specific date and time.

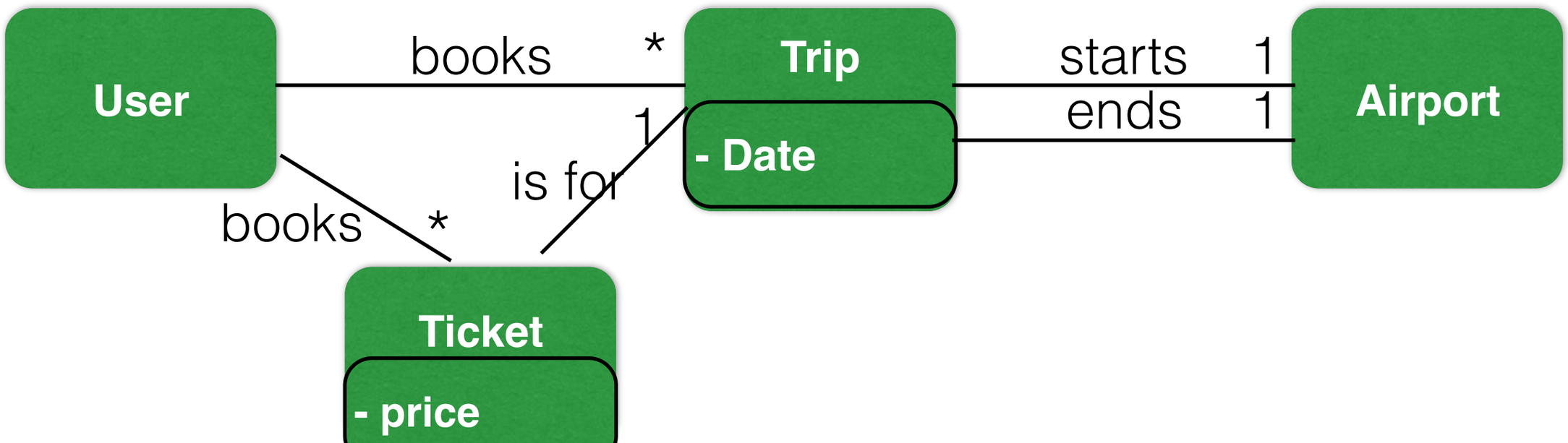


You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

- A user can book multiple one-way trips
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight.

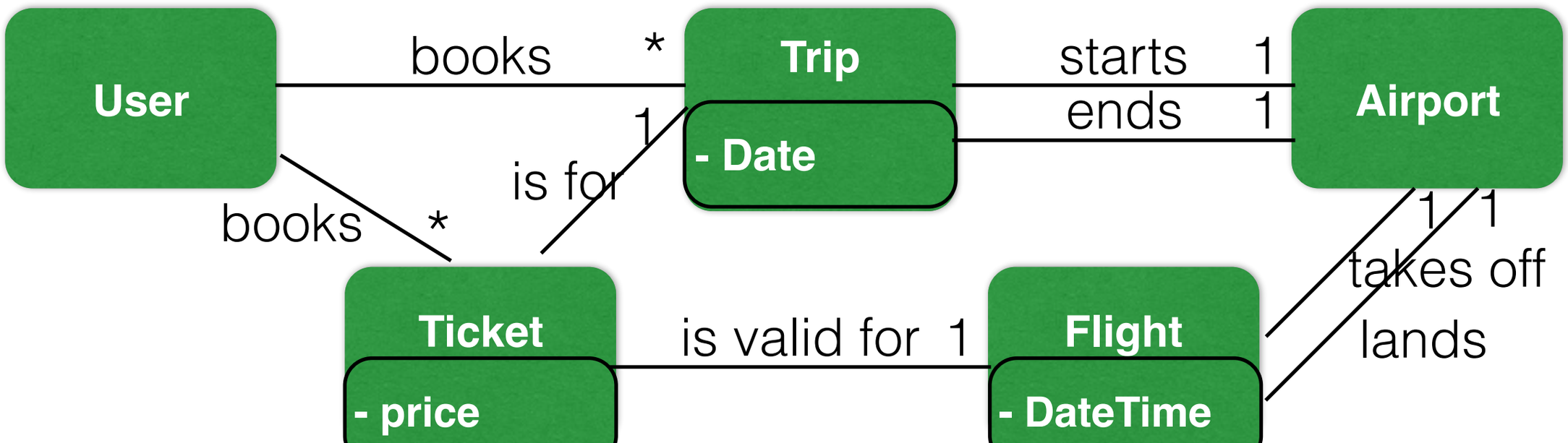
Flights takeoff and land in an airport and occur on a specific date and time.



You are asked to develop a website for searching, comparing and purchasing flights operated by third party airline companies.

Your system must satisfy the following requirements:

- A user can book multiple one-way trips
- Each trip occurs on a specific date. It starts and ends in two (different) airports.
- For each trip, the user needs to book one or more plane tickets. Tickets have a price (expressed in CHF) and are valid for a specific flight. Flights takeoff and land in an airport and occur on a specific date and time.



Exercise

Quality attributes: Fill out the following table. In each row, specify if the corresponding attribute is internal/external, if it applies to product/process/resources, and which metrics could be used for its measurement. 2 points

Quality Attr.	category*	application**	metrics
Reliability			
Productivity			

category* : external or internal
application**: product, process or resources

Quality attributes: Fill out the following table. In each row, specify if the corresponding attribute is internal/external, if it applies to product/process/resources, and which metrics could be used for its measurement. 2 points

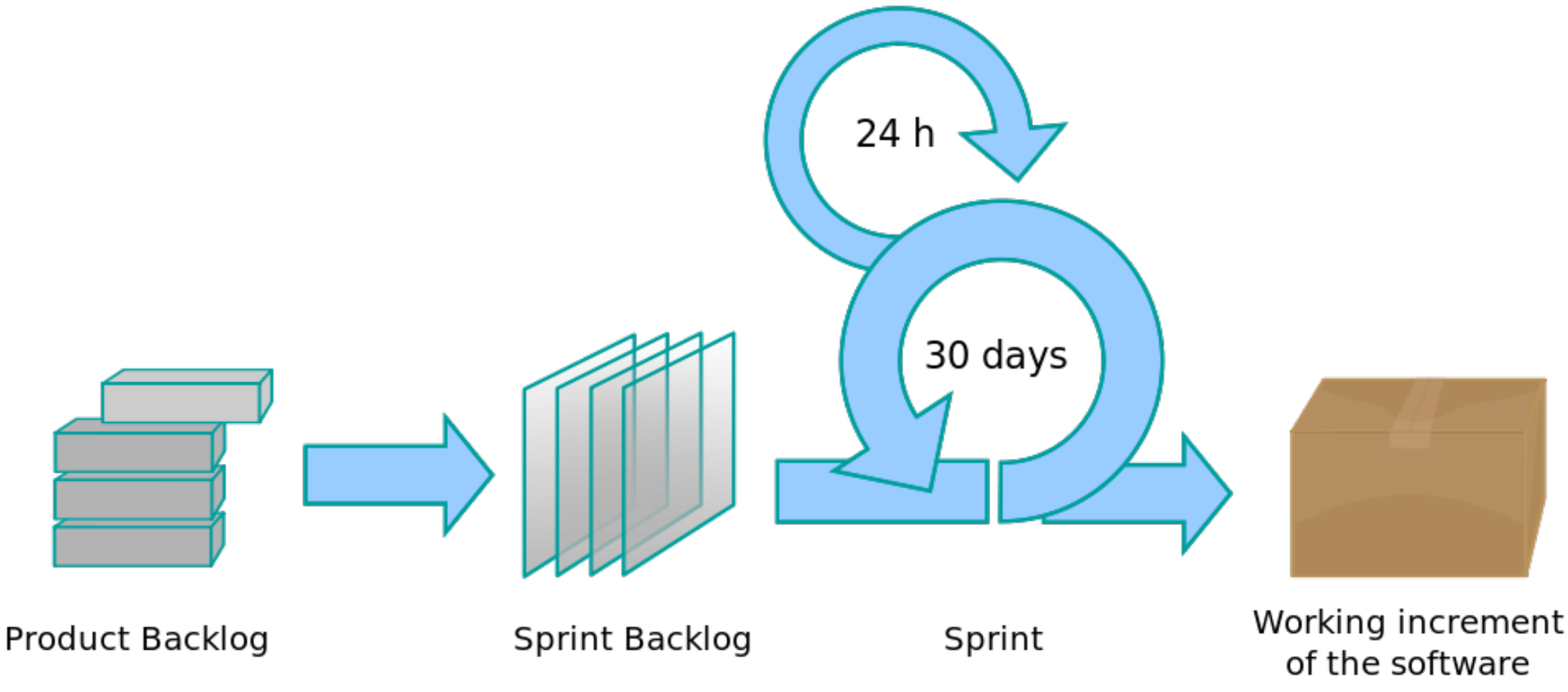
Quality Attr.	category*	application**	metrics
Reliability	Ext	Product	mean time between failure number of software faults
Productivity	Ext	Process	delivered use cases / month function points

category* : external or internal
application**: product, process or resources

Topics

- Terminology
- Software design/quality (principles & diagrams)
- **Software Engineering Processes**
- Software architecture (styles & properties)
- Testing (methods & techniques)

Software Engineering Processes



Software Engineering Processes

Product backlog (PBL)
A prioritized list of high-level requirements.

Sprint backlog (SBL)
A prioritized list of tasks to be completed during the sprint.

Sprint
A time period (typically 1–4 weeks) in which development occurs on a set of backlog items that the team has committed to. Also commonly referred to as a Time-box or iteration.

Increment
The sum of all the Product Backlog items completed during a sprint and all previous sprints.

Software Engineering Processes

Product Owner
The person responsible for maintaining the Product Backlog by representing the interests of the stakeholders, and ensuring the value of the work the Development Team does.

Scrum Master
The person responsible for the Scrum process, making sure it is used correctly and maximizing its benefits.

Development Team
A cross-functional group of people responsible for delivering potentially shippable increments of Product at the end of every Sprint.

Topics

- Terminology
- Software design/quality (principles & diagrams)
- Software Engineering Processes
- **Software architecture** (styles & properties)
- Testing (methods & techniques)

Software architecture

The **Design Structure Matrix** (DSM) is a simple, compact and visual representation of a system or project in the form of a matrix.

The off-diagonal cells are used to indicate relationships between the elements.
Reading across a row reveals what other elements the element in that row provides outputs to, and scanning a column reveals what other elements the element in that column receives inputs from.

Exercise

Software architecture

	A	B	C	D
A	–	15	0	0
B	0	–	18	0
C	0	0	–	13
D	0	0	0	–

- draw as package diagram
- which architectural style ?

	M	V	C
M	–	5	0
V	7	–	8
C	6	5	–

- is MVC correctly implemented ?

Topics

- Terminology
- Software design/quality (principles & diagrams)
- Software Engineering Processes
- Software architecture (styles & properties)
- **Testing** (methods & techniques)

Testing

```
1  public interface BankSystemInterface {
2    /** Registers a transaction from a sourceAccount to a destinationIBAN for a given amount.
3    The operation throws an exception if:
4         - sourceAccount is null
5         - destinationIBAN is null, an empty string, or an invalid IBAN code
6         - amount <= 0
7         - amount > 'balance of sourceAccount'  */
8    public void outboundMoneyTransfer(BankAccount sourceAccount, String targetIBAN, double amount);
9
10   /** Prepares cumulative bank-to-bank transactions based on all transactions registered on the ↩
         current day. Runs once a day. */
11   public void batchProcessTransactions();
12 }
```

1. Write test cases for this method: **outboundMoneyTransfer**(..).
   Make sure you cover all the error cases described in the javadoc comment.
   You don't need to write code. A Test case can be described as a set of inputs and an expected output.

   example (TestCase 0):
   - input: outboundMoneyTransfer(a,b,0): where a is .. and b is ..
   - output: Operation fails 2 points

- tests for achieving full branch coverage ?

Exercise

```
1  public class BankSystem implements BankSystemInterface {
2    Vector<OutboundTransaction> currentDayTransactions = new Vector<OutboundTransaction>();
3
4    public void outboundTransfer(BankAccount sourceAccount, String targetIBAN, double amount) {
5      if(sourceAccount == null || targetIBAN == null) {
6        throw new TransactionEx("Accounts cannot be null");
7      }
8      if(targetIBAN.isEmpty()) {
9        throw new TransactionEx("Target account not specified");
10     }
11     if(!this.isValidIBAN(targetIBAN)) {
12       throw new TransactionEx("Target account is not valid");
13     }
14     if(amount <= 0) {
15       throw new TransactionEx("Transaction amount is not valid");
16     }
17     if(amount > sourceAccount.balance) {
18       throw new TransactionEx("Source account has insufficient balance");
19     }
20
21     currentDayTransactions.add(new OutboundTransaction(sourceAccount, targetIBAN, amount));
22   }
23
24   public void batchProcessTransactions(){
25     Set<Bank> allBanks = new HashSet<Bank>();
26     for(OutboundTransaction t : currentDayTransactions){
27       Bank bank = this.getBankFromIBAN(t.targetIBAN);
28       allBanks.add(bank);
29       bank.credit = bank.credit + t.amount;
30       t.sourceAccount.balance = t.sourceAccount.balance - t.amount;
31     }
32     for(Bank bank : allBanks){ bank.claimCredit();  }
33   }
34
35   private Bank getBankFromIBAN(String IBAN){ .. }
36   private boolean isValidIBAN(String IBAN){ .. }
37 }
```

- tests for achieving full branch coverage ?

Exercise

```
1  public class BankSystem implements BankSystemInterface {
2    Vector<OutboundTransaction> currentDayTransactions = new Vector<OutboundTransaction>();
3
4    public void outboundTransfer(BankAccount sourceAccount, String targetIBAN, double amount) {
5      if(sourceAccount == null || targetIBAN == null) {
6        throw new TransactionEx("Accounts cannot be null");
7      }
8      if(targetIBAN.isEmpty()) {
9        throw new TransactionEx("Target account not specified");
10     }
11     if(!this.isValidIBAN(targetIBAN)) {
12       throw new TransactionEx("Target account is not valid");
13     }
14     if(amount <= 0) {
15       throw new TransactionEx("Transaction amount is not valid");
16     }
17     if(amount > sourceAccount.balance) {
18       throw new TransactionEx("Source account has insufficient balance");
19     }
20
21     currentDayTransactions.add(new OutboundTransaction(sourceAccount, targetIBAN, amount));
22   }
23
24   public void batchProcessTra
25     Set<Bank> allBanks = new
26     for(OutboundTransaction t
27       Bank bank = this.getBank
28       allBanks.add(bank);
29       bank.credit = bank.cred
30       t.sourceAccount.balance
31     }
32     for(Bank bank : allBanks)
33   }
34
35   private Bank getBankFromIBA
36   private boolean isValidIBAN
37 }
```

1. outboundTransfer(null, *any*, *any*)
2. outboundTransfer(sa, *“”*, *any*)
3. outboundTransfer(sa, *“000”*, *any*)
4. outboundTransfer(sa, *“1234…..1234”*, -1)
   where *“1234…..1234” is valid iban*
5. outboundTransfer(sa, *“1234…..1234”*, N)
   where N > sa.balance
6. outboundTransfer(sa, *“1234…..1234”*, N)
   where N <= sa.balance

Good Luck !