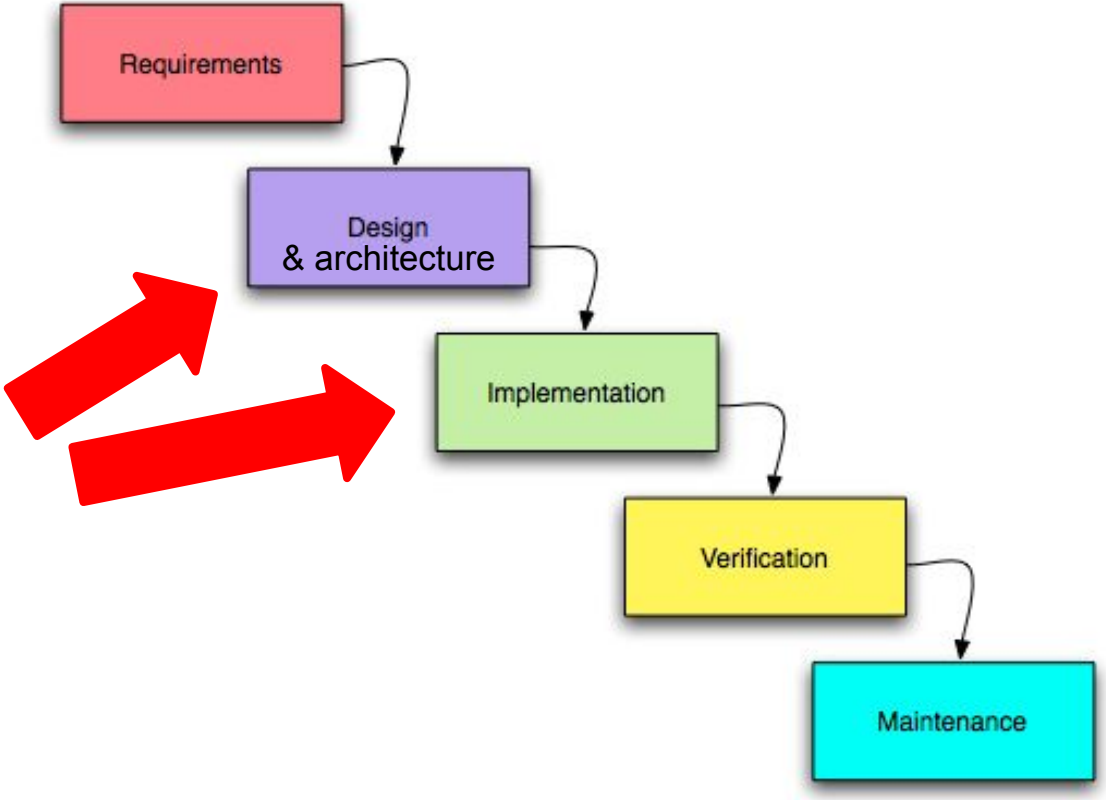
ORM
Object-relational mapping

Jana Kostičová, 15.11.2023

SDLC - Waterfall



Object-relational mapping - ORM

= a programming technique used to facilitate the interaction between object-oriented code and relational databases

ORM allows developers to work with databases using object-oriented programming (OOP) concepts, such as classes, objects, and methods, rather than writing raw SQL queries.

- Modeling data
- Creating / altering DB according to the model
- CRUD - inserting / querying / updating / deleting data

```
# conn = sqlite database connection
cursor = conn.execute("SELECT * from USER_ACCOUNT where
NAME='spongebob' or NAME='sandy'")

for row in cursor:
   print("ID = ", row[0])
   print("NAME = ", row[1])
   print("FULLNAME = ", row[2], "\n")

conn.close()
```

Traditional DB access

```
# engine = sqlalchemy db engine
# User = mapped class, part of the model
session = Session(engine)
stmt = select(User).where(User.name.in_(["spongebob", "sandy"]))

for user in session.scalars(stmt):
   print("ID = ", user.id)
   print("NAME = ", user.name)
   print("FULLNAME = ", user.fullname, "\n")

session.close()
```

ORM

ORM - pros and cons

(+) **Abstraction of database details** - ORM abstracts the low-level DB details and allows developers to work with high-level, object-oriented code, making it easier to understand and maintain
(+) **OO approach** - ORM aligns well with object-oriented programming (OOP) principles. It allows you to model your data as objects, which can make your code more intuitive and maintainable
(+) **Reduced amount of code** - the amount of repetitive SQL code and database-related logic is reduced

(-) **Abstraction of database details** - ORM abstracts DB details, which can limit the control over certain database-specific features and optimizations.
(-) **Possible performance overhead** - queries are created and executed dynamically and cannot be hand-optimized
(-) **Steeper learning curve**
(-) **Complex queries** - it may be difficult to express more complex queries

Object-relational impedance mismatch

There are several fundamental differences between OO design and DB design:

- Classes have instances, inheritance, relationships; relational databases have just tables
- References vs pointers
- Datatype differences
- Database normal forms make little sense in OOP

Object-relational impedance mismatch is a set diculties that encountered if we use relational databases with an OO application.

OODBMSs and document stores

OODBMS - Object-oriented databases

- Eliminate the need for converting data to and from its SQL form, as the data is stored in its original object representation and relationships are directly represented

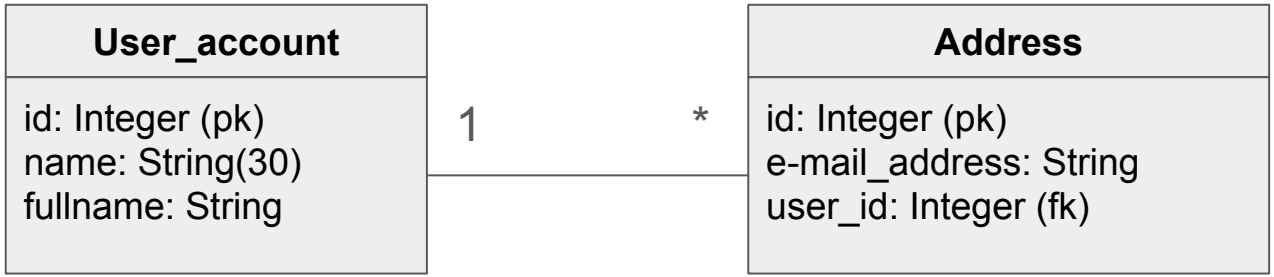
Documents stores (JSON, XML)

- Storing objects in tree structures is more straightforward than in relational tables

ORM software - examples

- Java: Hibernate
- Python: SQLAlchemy, Django ORM
- C# / .NET: Entity Framework
- JavaScript / TypeScript / Node.js: Sequelize, TypeORM, Prisma
- …

Example - DB model



Resources & further reading

- Robert Lukotka: Objects and databases, ORM
- Wikipedia - Object–relational mapping
- SQLAlchemy 2.0 Documentation
- Python documentation: typing — Support for type hints