ML and Numerical Software Development

ML Software Development-I

Organon Analytics

December 17, 2019

Agenda

- Preparations
- Why do we develop software anyway?
- Common concerns in numerical software
- Software Architecture
- Some pearls of architectural wisdom
- Layered Architecture
- GBM software: Main computation flow

Preparations

- Download&install Visual Studio 2019
- Open an account at GitHub. Read GitHub tutorial. Learn Git
- Download&Install PostgreSQL
- Import data into PostgreSQL
- Install DBeaver to access data in PostgreSQL

See the link for a tutorial: `https://github.com/Organon-Analytics/ML-Course/tree/master/Documents`

Resources

- Preps: `https://github.com/Organon-Analytics/ML-Course/tree/master/Documents`
- GBM pseudo-code: `https://github.com/Organon-Analytics/ML-Course/tree/master/References`. "XGBoost - A GBM Algorithm.pdf"
- *Architecting Applications for the Enterprise 2nd Edition, by Dino Esposito et al.*
- *Refactoring, Improving the Design of Existing Code by Martin Fowler et al.*
- *Code Complete 2, by Steve McConnell*
- *The Mythical Man-Month, Essays on Software Engineering by Fred Brooks*
- A C# tutorial

Why do we develop software anyway?

The business need: A production-grade GBM algorithm you can deploy in a commercial environment

1. Open Source: There are various OS implementations of GBM:
   - Spark-ML
   - Scikit-Learn
   - TensorFlow
   - XGBoost
   - LightGBM
   - CatBoost
2. In-house: Develop one of your own

There seems to be many options. Why do you develop yet another software?

Why in-house?

If an open source software

1. Meets your MUST-HAVE functional requirements
2. Meets your MUST-HAVE non-functional requirements
3. Has stood the test of time: Tested and used by a large enough community
4. Has good documentation
5. Has little or no difficulty regarding integration with your current software

Then use it. Otherwise develop your own

Open source software is good in theory and on paper but you might have some problems $\Longrightarrow$

Why in-house?

Cons of using open-source software

1. Lack of important features. e.g. the hyper-parameter optimization in current GBM implementations
2. Not easy to extend. Most often, codes are hard to understand and hard to extend
3. Fails under extremities: e.g. if you need to build thousands of models, or build models on thousands of columns, the OS algorithms fail. They are usually built to endure "normal conditions", not stand the storms
4. Not fast enough: Optimizing numerical code takes an orders of magnitudes more effort than simply developing it. Most of the algorithms are not optimized

So, all in all, it is an optimization problem:
if you demand better software (more accurate, faster, open for extension) and if you can meet the costs, go for it

Common concerns in numerical software

- Database
- Cache optimization
- Exception handling
- Parallelism
- Distribution
- Check-pointing
- High performance matrix computations
- Dense vs. sparse computations
- Job scheduling

Software Architecture

- Structure
- Organization
- Plan
- Design
- Style
- Templates, patterns
- Build
- Artifacts
- Standardize

Some pearls of architectural wisdom

- ⋆ Good judgment comes from experience, and experience comes from bad judgment
- ⋆ Do not fear of being mistaken. Be fearful of not being bold enough
- ⋆ Benefit from others' experiences. Enhance it with your own
- ⋆ The best design is a mixture of exploitation and exploration. The ratio is a hyper-parameter. Find it for yourself
- ⋆ Software is closer to a sculpture rather than a bridge. It is mostly art, partially engineering
- ⋆ Design for simplicity. Period.
- ⋆ Software does not need to be complex per se. Complexity is an illusion and reflect a lack of knowledge&experience. Your design get simpler as you age.

Sanjay&Jeff

"...Their code's too loose. One screen of code has very little information on it. You're always scrolling back and forth to figure out what's going on." Others write code that's too dense: "You look at it, you're, like, 'Ugh. I'm not looking forward to reading this.' Sanjay has somehow split the middle. You look at his code and you're, like, 'O.K., I can figure this out,' and, still, you get a lot on a single page." Silverstein continued, "Whenever I want to add new functionality to Sanjay's code, it seems like the hooks are already there. I feel like Salieri. I understand the greatness. I don't understand how it's done."

Layered Architecture

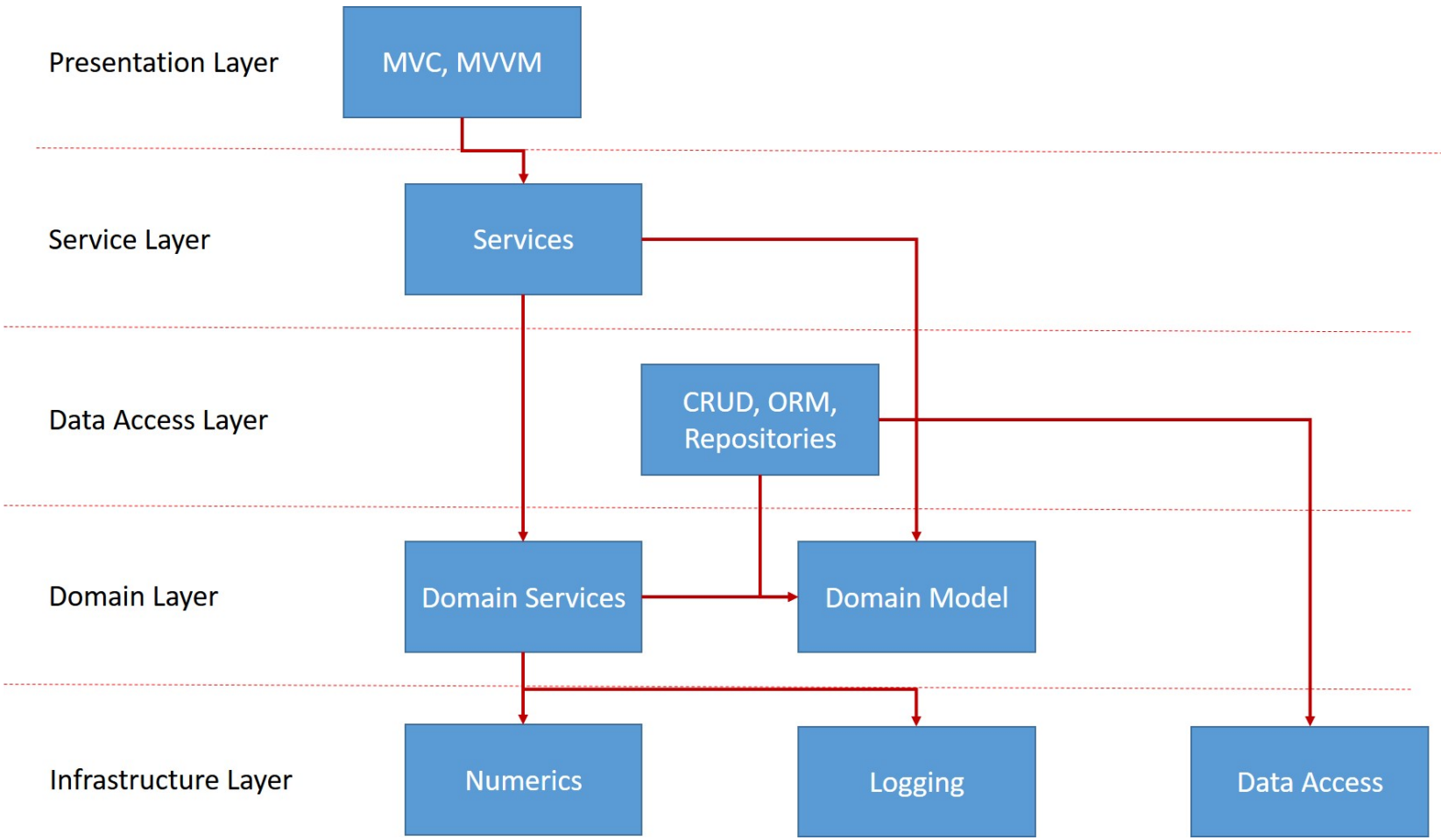
Primary goal in software development:
Manage the complexity: Find the optimal complexity where

1. Current requirements are fully met
2. Extension&maintenance is not impossible

The first technique in managing complexity is to reduce the dependencies at various levels:

- Dependency between modules
- Dependency between&within classes inside a module
- Dependency on rigid structures
- Dependency on external software

Layered Architecture



Presentation layer: Hosts UI classes. Uses services in the Service Layer to call system functionality. Well established patterns exist: MVM, MVVM. Learn them if you are involved in such development. Should depend only on Service Layer

Service Layer: The interface layer between the system and its consumers. Consumers supply the necessary inputs and related service(s) calls the code to implement the requested functionality. Should mainly depend on Domain Layer for system functionality, and Infrastucture for cross-cutting concerns

Domain Layer: Hosts the Model(data structures) and Service(algorithms) classes. The core of the system functionality sits here. Should depend only on Infrastructure for basic computations, e.g. numerics

Data Layer: Acts as the intermediary between system memory and the data on disk. Implements CRUD(Create-Read-Update-Delete) functionality. Executes the translation between the objects in memory and database objects on disk

Infrastructure Layer: Common functionality that is needed by the system, such as numerics, logging, validation, Inversion of Control, physical database access, etc.

Layered Architecture

Dependencies:

- Presentation $\rightarrow$ Service
- Service $\rightarrow$ Domain, Infrastructure
- Data $\rightarrow$ Domain, Infrastructure
- Domain $\rightarrow$ Infrastructure

6 direction of dependencies out of a possible 20

Reduced dependencies at the highest level

GBM software: Main computation flow

1. Read configuration file *(Optional]*
2. Validate settings(i.e. parameters, configurations)
3. Read data&metadata to populate primary data structures
4. Prepare/Optimize in-memory data structures
5. Run algorithm and produce results
6. Organize&save results
0. Validate input&output
0. Throw exceptions
0. Parallelize
0. Save runtime statistics
0. Log messages

Steps numbered with "0" are scattered everywhere in the program body