

ES Software Engineering

Lecture 7

System and Data architecture design Software licence

In the previous lecture

1. User interface design
2. Testing
3. Implementation – decreasing of an abstract level
4. Framework vs. Libraries
5. Documentation
6. Reverse engineering

Plan of the lecture

1. System architecture design
2. Data structure design – files vs. databases; databases comparison
3. Agile programming
4. Open Source
5. Intellectual Property
6. Software Licences

System architecture design

System architecture types:

1. Integrated architecture
2. Distributed architecture
 - server based
 - client based
 - client-server
 - peer to peer

System architecture design

Server types:

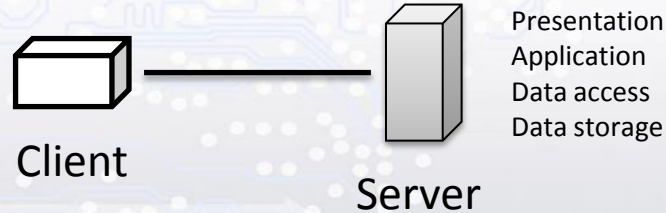
- Mainframe
- Minicomputer
- Microcomputer

Client types

- Personal computer (desktop, laptop, tablet)
- Terminal
- Special terminal (ATM machine, info-kiosk, smartphone, smartwatch)

System architecture design

Server based architecture



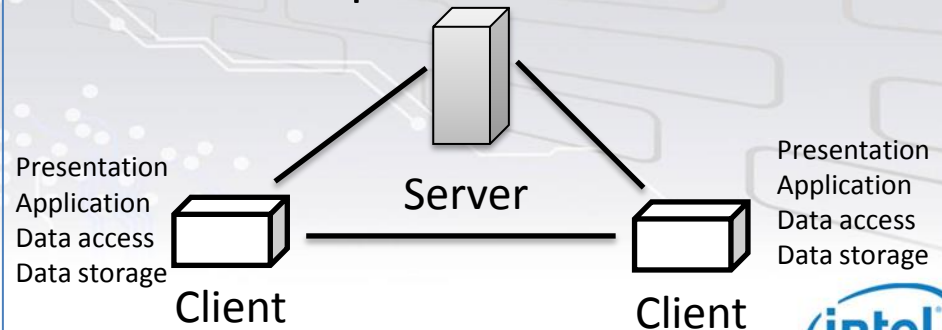
Client based architecture



Client-server architecture



Peer to peer architecture



Data structure design

Data storage strategy:

1. Files

- Sequenced files
- Random access files

2. Databases

- Relational (SQL)
- NoSQL (not only SQL)
- Object-Oriented

Data structure design

Ad. 1. File-based data storage

Pros:

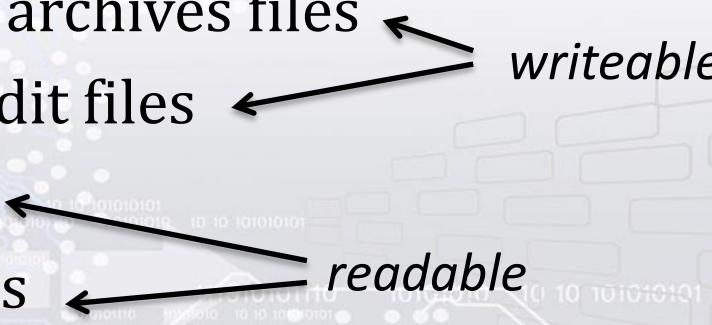
- Flexibility
- Any type and format
- High efficiency reading and writing
- Easy access

Cons:

- No sharing mechanism and no modification mechanism
- Weak access control mechanism
- Data redundancy

Data structure design

Ad. 1. File-based data storage using area

- History and archives files
 - Log files, audit files
 - Config files
 - Look-up files
 - Transact files
- writeable*
- readable*
- 

Data structure design

Ad. 2. Database storage using area

- Systems of systems architecture
- Multi-user systems
- Transaction-based systems
- Management info systems
- Expert systems (interdisciplinary)

Data structure design

Ad. 2. Database storage using area

- Systems of systems architecture
- Multi-user systems
- Transaction-based systems
- Management info systems
- Expert systems (interdisciplinary)

Data structure design

Ad. 2. Database Management System (DBMS):

- RDBMS – Relational Database Management System (SQL access)
- NoSQL – Not Only SQL Database – XML, graphic data
- ORDBMS – Object-Relational Database Management System
- OODBMS – Object-Oriented Database Management System

Data structure design

RDBMS

Pros:

- Fast data access
- Standard access language (SQL)
- Multiuser access
- Data consistency checking
- Widely used

Cons:

- No complex data types and lack of full support for simple data types (i.e. int64)
- Lack of support for object-oriented data identification and inheritance

Data structure design

RDBMS

Pros:

- Fast data access
- Standard access language (SQL)
- Multiuser access
- Data consistency checking
- Widely used

Cons:

- No complex data types and lack of full support for simple data types (i.e. int64)
- Lack of support for object-oriented data identification and inheritance

Data structure design

ORDBMS

Pros:

- Support for complex data types
- Standard access language (SQL)
- Multiuser access
- Data consistency checking

Cons:

- Lack of full support for simple data types (i.e. int64)
- Lack of support for object-oriented data identification and inheritance
- Not well known and not widely used

Data structure design

OODBMS

Pros:

- Support for complex data types
- Support for Object-Oriented data identification
- Multiuser access
- Data consistency checking

Cons:

- Lack of standard access
- Multi-OO-language support
- Not well known and not widely used

Agile programming

Agile programming circumstance – weakness of the classic approach

- Long time for concrete results
- Problem of specifying requirements at the early project stage
- Requirements meeting is not guaranteed
- Developers loose a contact with clients for a long time

Agile programming

Agile Manifesto (Manifesto for Agile Software Development) – declaration of common principles for agile software development (2001 r.):

- Extreme programming (XP)
- SCRUM
- Dynamic Systems Development Method (DSDM)
- Adaptive Software Development
- Feature Driven Development (FDD)
- Pragmatic Programming

Agile programming

Agile Manifesto

individuals
interactions

working
software

customer
collaboration

responding
to changes

Agile level

processes,
tools

comprehensive
documentation

contract
negotiation

following
a plan

Agile programming

XP

- ... you don't know exactly what you are doing
- you also don't know how to make it the best way
- concerns smaller or medium-sized projects
- but most of all high-risk projects

Agile programming

XP – what are the recommendations?

- „*release early, release often*” – frequent iterations, only the closest iteration is planned
- not designed in advance because the system architecture is unknown
- first testing, then coding
- Programming in pairs – change at about 1.5h, which ensures the readability of code and elimination of errors

Agile programming

SCRUM – agile product development methodology (1986). Since 1995 used in software development.

- Process divided into sprints (short iterations)
- Sprint lasts 1-4 weeks
- Prerequisites collected in the form of stories
- Short daily meetings (15 min)
- Product Backlog & Sprint Backlog

Agile programming

Testing in agile programming

- Each new or modified code must not be placed in repository until it passes all tests
- Tests are designed basing on user stories.
- One story may have many tests.
- Implementation of each story is finished when it passes all tests.

Agile programming

Agile programming – summary

- Used for simple applications
- Low risk at project failure
- Time consumed only on development and testing
- Schedule with frequent release
- No need of defined requirements
- Close cooperation between customers and developers during the whole time of working on the project
- Tests defined by users
- Process dependant on changes in people and team
- Lack of documentation – (unreadable project after time)

Open Source

Open Source – approach to software development which allows access to source code or other results of somebody's work.

1998 – the beginning of movement initiated by several IT specialists and hackers

Names associated with the movement: John Maddog Hall, Eric Raymond, Bruce Perens, Linus Torvalds, Larry Wall, Guido van Rossum. A fraction of the movement is free software, primarily associated with Richard Stallman.

Open Source and Free Software movements are related to numerous controversies. Also views of the creators and members of councils of development of both movements are controversial.

Open Source

Does it pay off?

Most open source software is nowadays created in commercial companies [8].

I. Embedded Systems business model:

*We can give
you for free:*



If you need this:

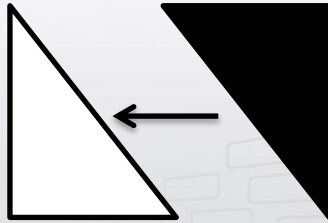
*We can make it
for you for pay.*



Open Source

II. Embedded Systems business model:

*We have always
given you free
software:*



*In our offer you
can find
hardware for
pay.*

*Of course others offer you the same hardware, but do you really
want to learn using other software?*

III. Embedded Systems business model:

Paid support and documentation (e.g. Red Hat, Novell, Cedega)

Open Source

*„all software free of
intellectual property restrictions...”*

*„... because these restrictions work against technical improvement
and the good of the community” (Free Software Foundation)*



*„... because of economic and technical aspects – not moral or ethical”
(Open Source Initiative)*



Open Source

Four freedoms:

Freedom 0 – run the program for any purpose

Freedom 1 – access to source code (analysis how software works and using it for your needs)

Freedom 2 – redistribute copies

Freedom 3 – improve the software and release improvements (community benefits)

Open Source

Intellectual Property?

„Intellectual property (IP) refers to creations of the mind, such as inventions; literary and artistic works; designs; and symbols, names and images used in commerce.

IP is protected in law by, for example, [patents](#), [copyright](#) and [trademarks](#), which enable people to earn recognition or financial benefit from what they invent or create. By striking the right balance between the interests of innovators and the wider public interest, the IP system aims to foster an environment in which creativity and innovation can flourish.”

source: World Intellectual Property Organization <http://www.wipo.int/about-ip/en/>

Software Licences

Software Licences – a legal instrument governing the usage or redistribution of copyright protected software.

Software Licence Categories:

- Proprietary licences
- Free software licences
- Open Source licences

Software Licences

Proprietary Licences

- Software publisher grants a license to use, copy or modify software
- Ownership of software copies remains with the software publisher
- All rights regarding the software are reserved by publisher
- Licence acceptance is a necessary condition of using a software
- Example: Microsoft Windows

Software Licences

End-User License Agreement

- License is usually in a form of accession-adhesion
- It is distributed in a form of a Accept/Deny dialog box
- Used to exempt publisher from responsibilities for events caused by the software
- Problem of illegal hidden charges in terms of use

Software Licences

Free Software Licences

- Ownership of software copies doesn't remain with the software publisher
- Ownership of the copy is transferred to the end user
- Ownership of the copyright remains with the software publisher
- copy owner != copyright owner
- Acceptance of license is optional (end user may use software without licence acceptance)
- Licences in some cases can be extended to Software License
- Examples: GNU & GPL

Software Licences

Open Source License

1. Copyleft licence:

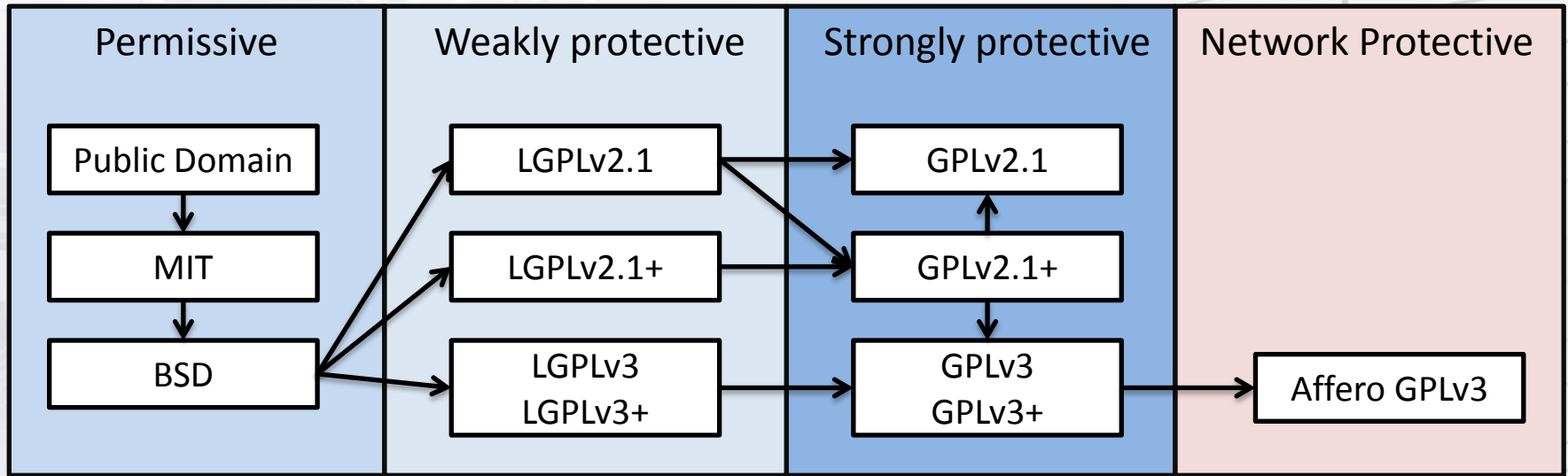
- gives the end user a permission to redistribute, reverse engineer or modify the software
- is not fully free of obligations, end-user must comply with certain terms
- example: GPL

2. Permissive licence

- permission to do anything with the source code
- no obligations
- can be used in proprietary software
- example: BSD, MIT

Software Licences

Popular software licences



Is based on A and B but has to be published on licence B

Software Licences

Public domain

- It isn't a licence but works like one.
- User can do anything in public domain and software has to be explicitly released to public domain.

MIT

- User can do anything with a code but can not sign it (claim that they wrote it)
- Non-copyleft licence, compatible with GPL, used for Xfree86 – popular X Window implementation for UNIX

BSD

- 4-clause version – using of authors names is forbidden in advertisement and is required in documentation
- 3-clause version – using of authors names is forbidden in advertisement
- 2-clause version – using of authors names is not considered

Software Licences

GPL

- Strong copyleft licence
- Code distributed under GPL license can not be used in programs distributed under other licences
- GPL grants users four basic freedoms
- Text of the licence is not under GPL – so the licence copyrights forbid modification of the licence
- Recipients have to get a copy of the licence with the program, if they want to copy or distribute the licence
- Users can modify the licence under 3 conditions: changing their names, not using „GNU” and removing the pre-ambule

Software Licences

Lesser GPL

- More permissive version of GPL
- Elaborated for software libraries
- Main difference is that LGPL allows to join programs which are not under GPL or LGPL – independently, if they are free or commercial
- In case of derivative work it allows modifications and reverse engineering for debugging them
- Since version 3 it is not a standalone licence and exists as a case of exception from GPL

Software Licences

Affero General Public Licence

- GPL-like license
- Elaborated for using for networking server software
- It expands using software over a network
- Source code must be available to any network user
- Used by operators who are not publishers of the software but who make it available in a network

Software Licences

Summary

To choose a licence you should consider a business model and then answer the following questions:

1. Do we want to allow commercial usage of our work?
2. Do we want to allow creation of derivative works?
3. If we allow creation of derivative work should it be released under the same licence?

Bibliography

- [1] Jarosław Kuchta, *System architecture design*, Embedded Systems Software Engineering, 2015
- [2] Roger S. Pressman: *Software Engineering. A Practitioner's Approach* (book, PDF)
- [3] Dennis A., Wixom B.H., Tegarden D., *Systems Analysis & Design. An Object-Oriented Approach with UML*, John Wiley and Sons, USA, 2002
- [4] Jarosław Kuchta, *Data structure design*, Embedded Systems Software Engineering, 2015
- [5] Jarosław Kuchta, *Agile Programming Methods*, Embedded Systems Software Engineering, 2015
- [6] Kent Beck: *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999
- [7] <http://www.agile.com>
- [8] Josh Lerner, Mark Schankerman: *The Comingled Code: Open Source and Economic Development*. Cambridge, MA: MIT Press, 2010
- [9] Tomasz Boiński, *Open Source*, Embedded Systems Software Engineering, 2015
- [10] World Intellectual Property Organization <http://www.wipo.int/about-ip/en/>
- [11] Free Software Foundatoin <http://www.fsf.org>
- [12] Open Source Initiative <http://www.opensource.org>
- [13] The Debian GNU Linux Project <http://www.debian.org/>