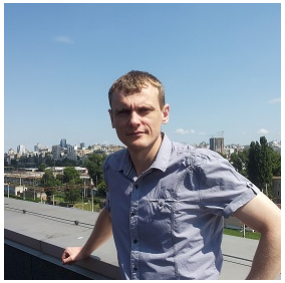
Yet Another Software Engineering Course

Hello!

My name is **Vladyslav Kurmaz**

You can find me at:
vladislav.kurmaz@gmail.com
vladyslav.kurmaz@globallogic.com
Linkein, Github, Facebook, G+



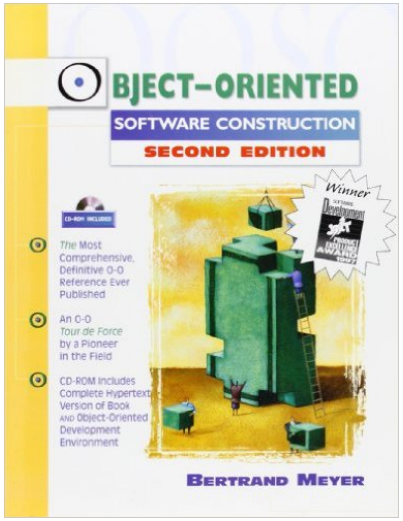
Key actors

Books 2 read


Бертран Мейер
Объектно-ориентированное конструирование программных систем
РУССКАЯ РЕДАКЦИЯ
ИНТЕРНЕТ УНИВЕРСИТЕТ


РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ
ПРИНЦИПЫ И ПАРАДИГМЫ
Э. ТАНЕНБАУМ, М. ван СТЕЕН
ПИТЕР

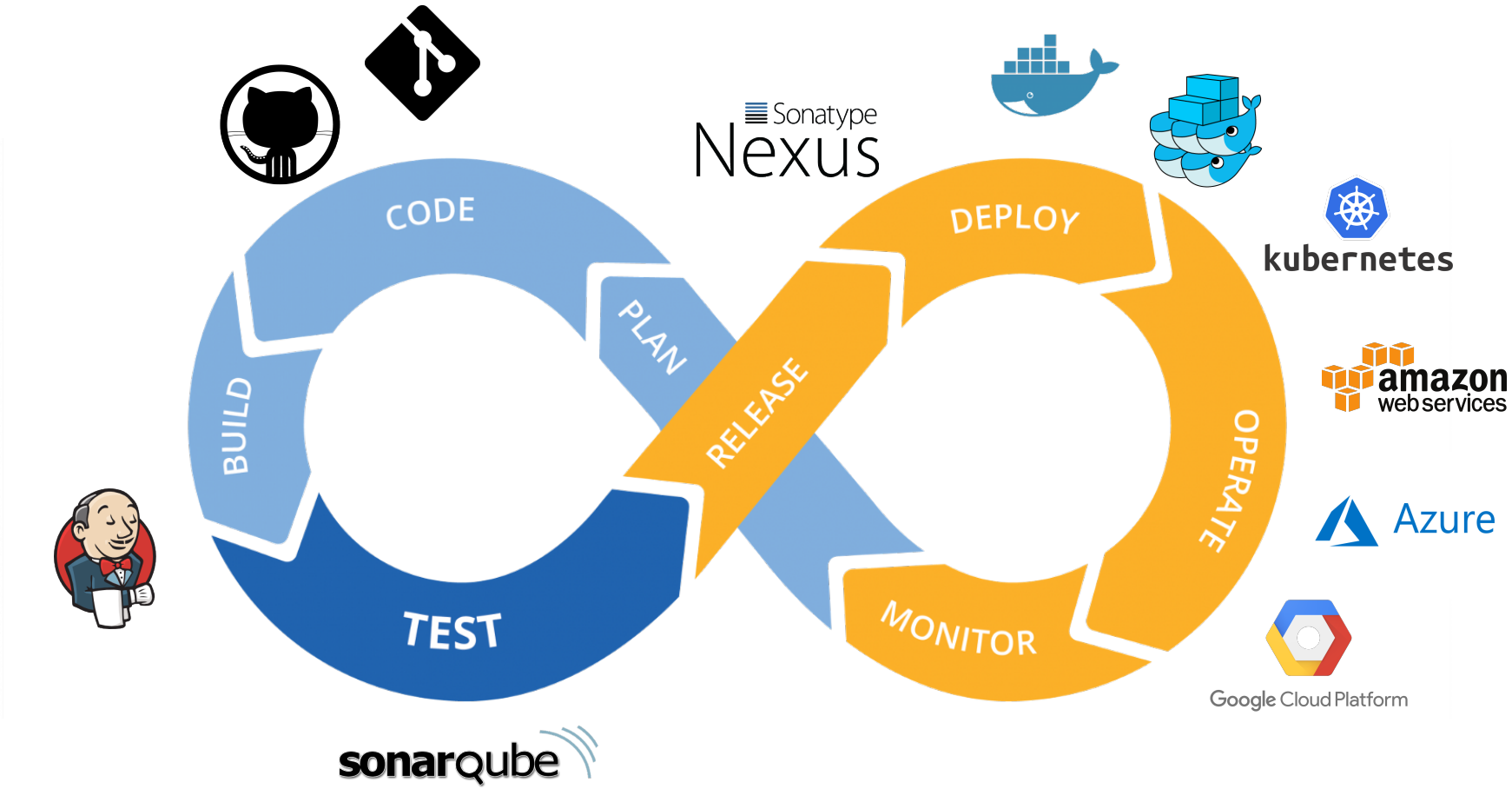

КЛАССИКА COMPUTER SCIENCE
СОВРЕМЕННЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ
4-е ИЗДАНИЕ
Э. ТАНЕНБАУМ
Х. БОС
ПИТЕР


BJECT-ORIENTED
SOFTWARE CONSTRUCTION
SECOND EDITION
Winner
The Most Comprehensive, Definitive O-O Reference Ever Published
An O-O Tour de Force by a Pioneer in the Field
CD-ROM includes Complete Hypertext Version of Book AND Object-Oriented Development Environment
BERTRAND MEYER

Challenges

- Programming vs Software development
- Dropbox-like application
- Be able to setup 0-time maintenance CICD development environment

Development pipeline



Software & Hardware development

- Common parts
  - Math, Finite State machine
  - Tools
  - Languages
  - OS
- Differences
  - Output: Executables(files) vs Executables+Physical stuff
  - Hardware development is more difficult from debug and testing perspective
  - Software is more easy to replicate

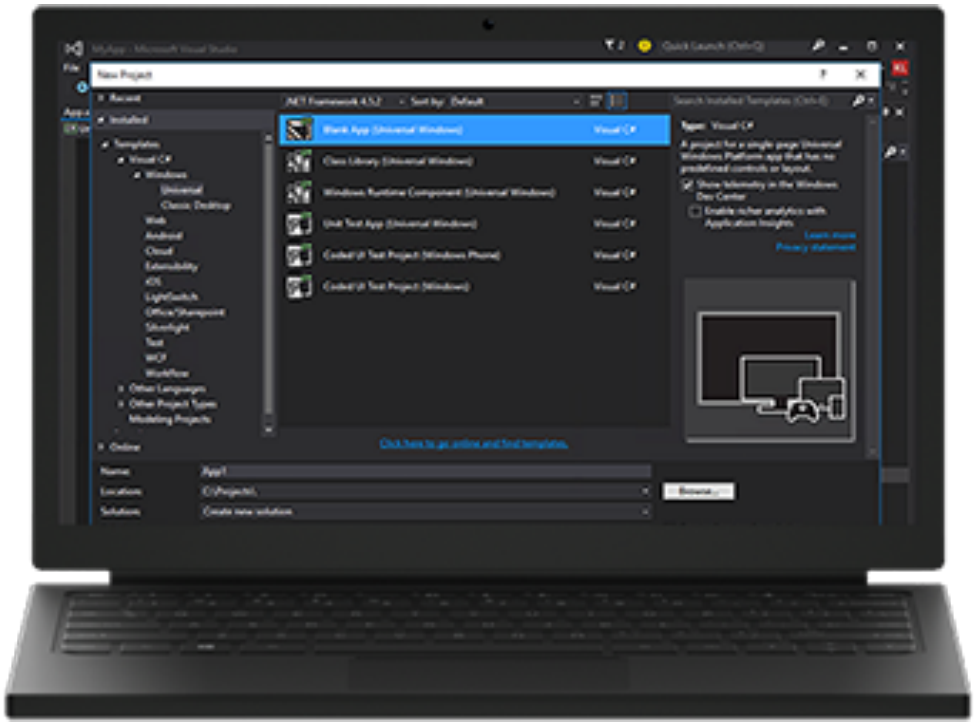
Roles

- Software developer
- Database developer
- Tester
- Business analyst
- Release manager
- DevOps
- Administrator
- Solution Architect

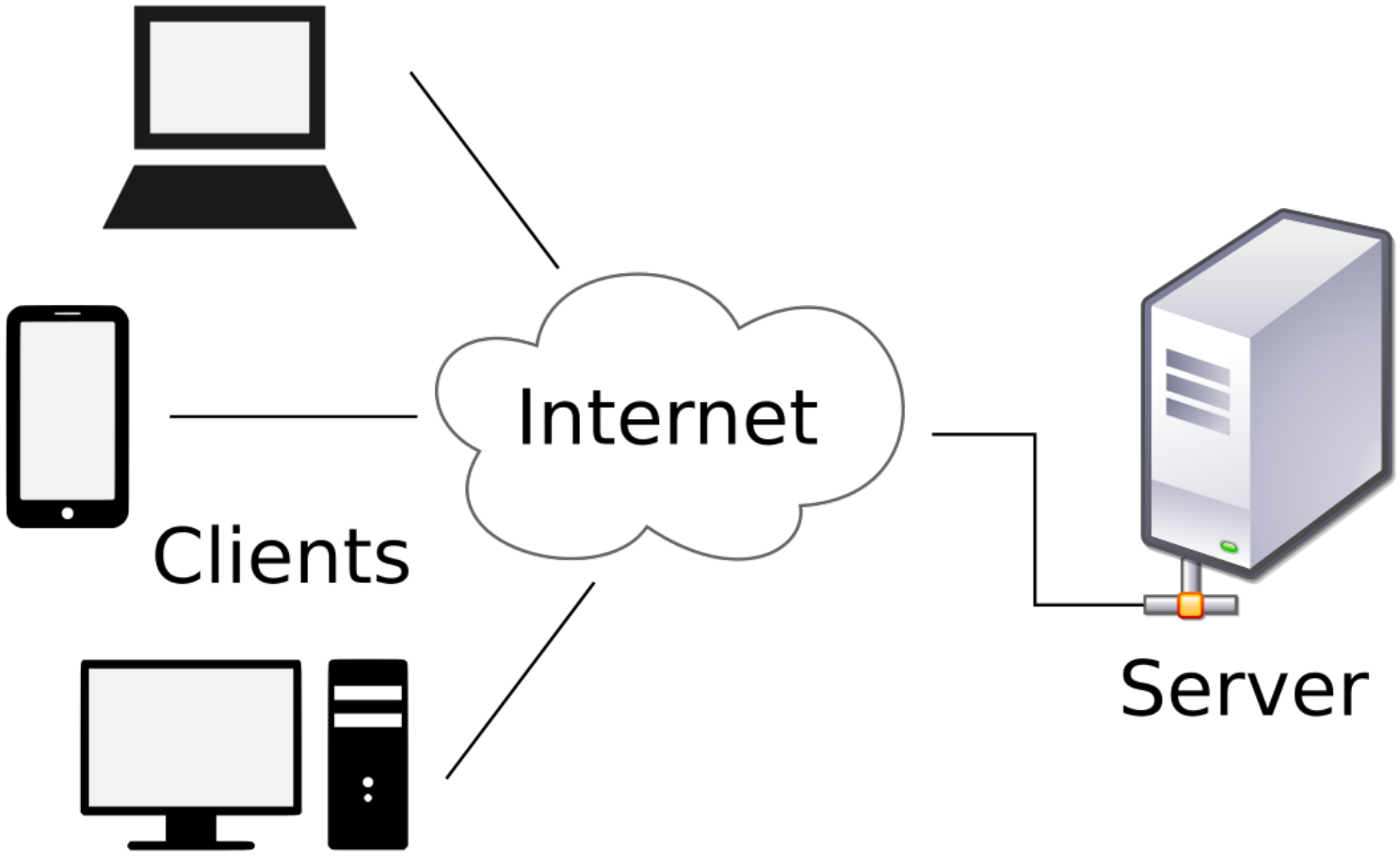
Technology Stacks

- Cloud provides, On-Premise, On-Demand
- OS – Windows, Linux, Hybrid
- Programming languages (Compilers & Interpreters, C/C++/Java/C#/PHP/Python/Go/D/Erlang)
- Data storages – RDB, GraphDB, NoSQL
- Frameworks, Ecosystems (Boost/Spring/Laravel, Java/.Net)
- Tools, libraries

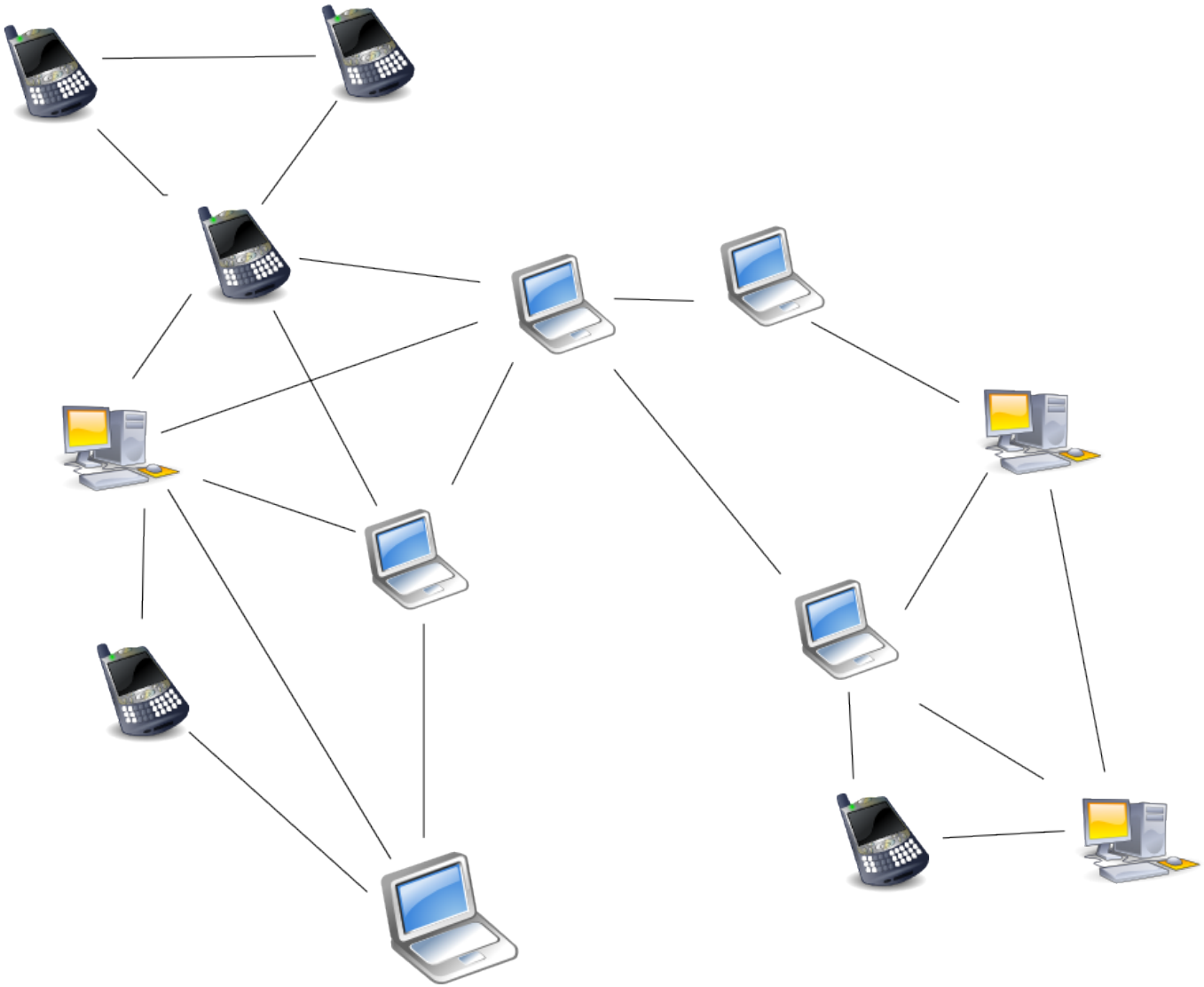
Types of Architecture – Standalone application



Types of Architecture - Client-Server application



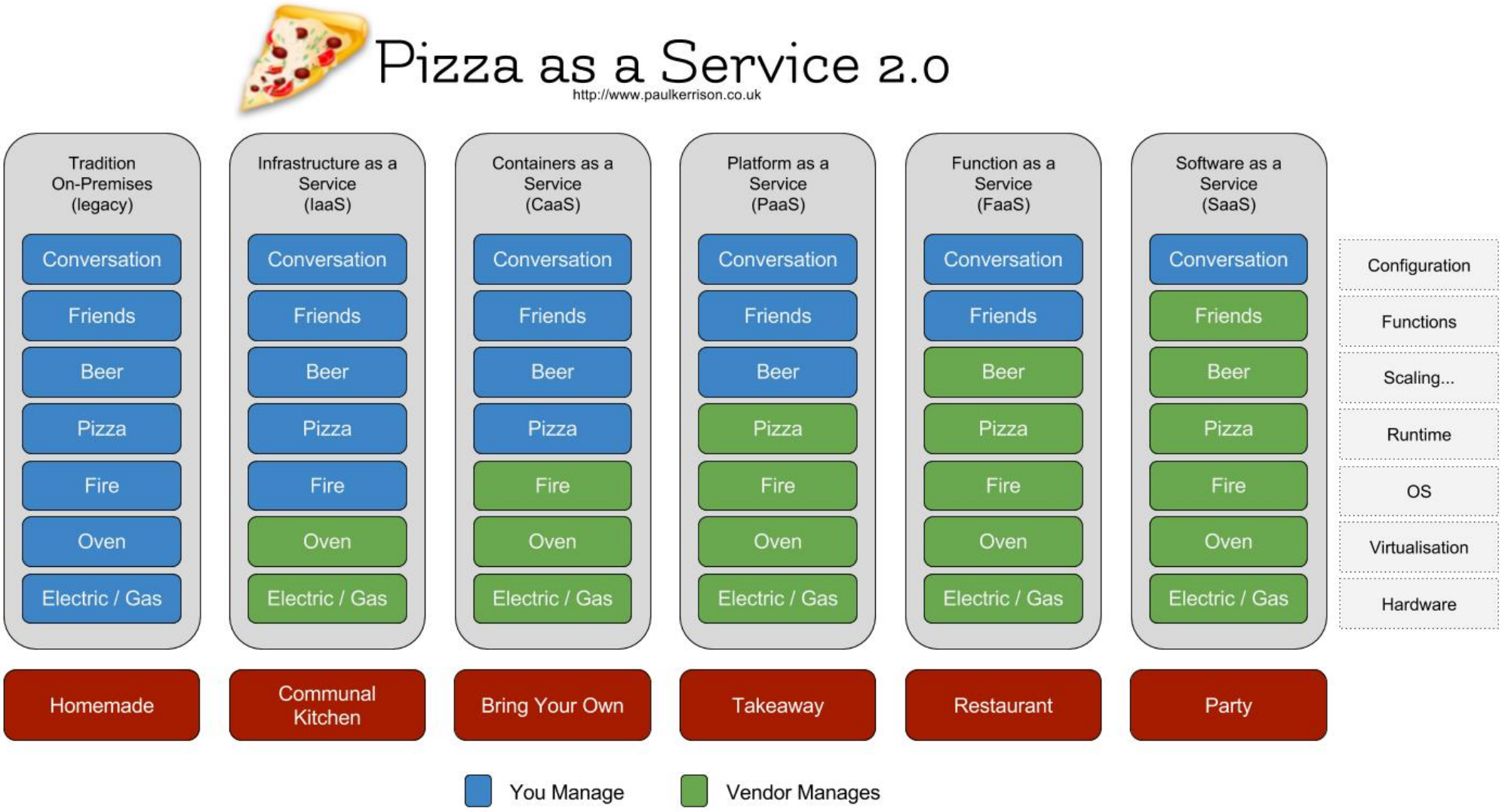
Types of Architecture – Peer-2-Peer



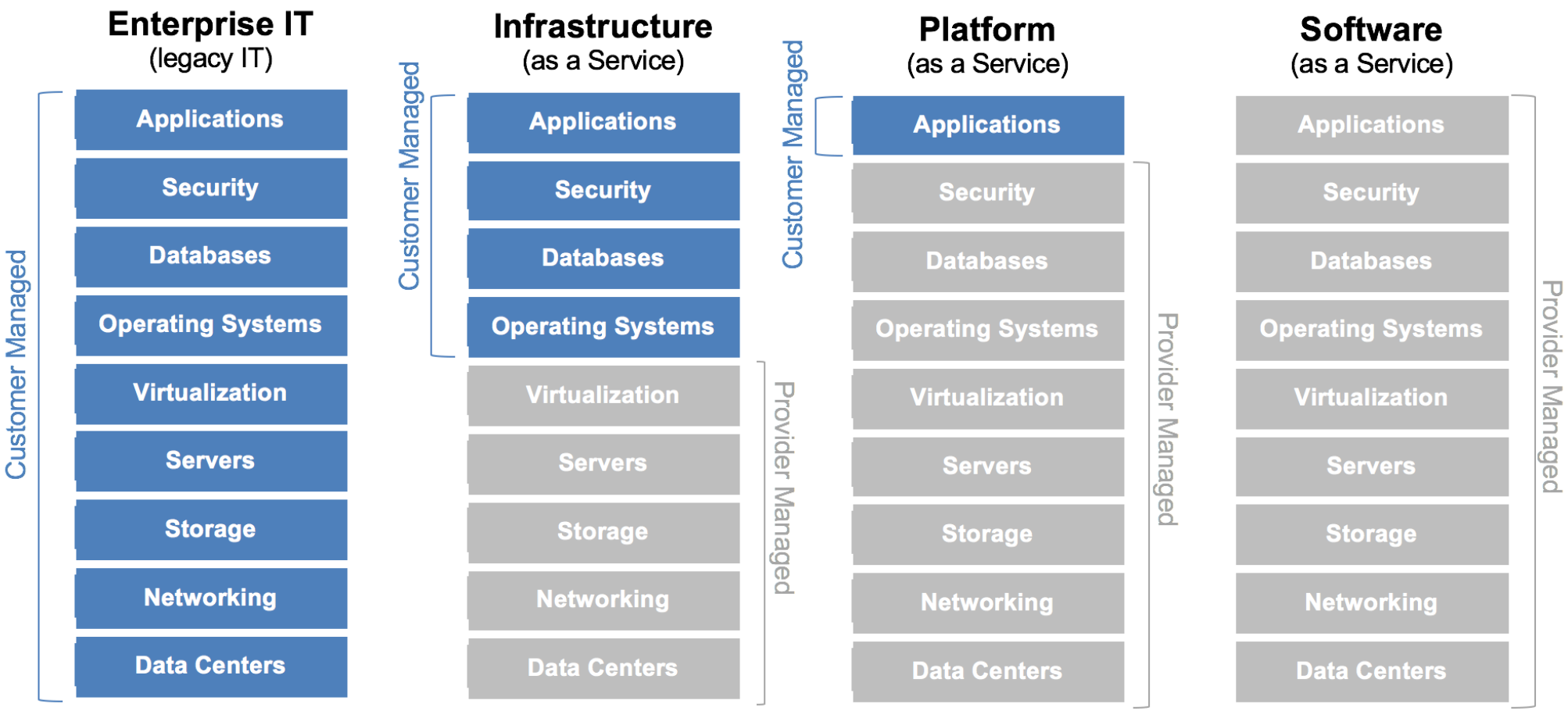
Types of Architecture - Monolith vs Micro services

Cloud is just someone else's computer

Clouds



Clouds


Enterprise IT
(legacy IT)
Applications
Security
Databases
Operating Systems
Virtualization
Servers
Storage
Networking
Data Centers
Customer Managed
Infrastructure
(as a Service)
Applications
Security
Databases
Operating Systems
Virtualization
Servers
Storage
Networking
Data Centers
Customer Managed
Provider Managed
Platform
(as a Service)
Applications
Security
Databases
Operating Systems
Virtualization
Servers
Storage
Networking
Data Centers
Customer Managed
Provider Managed
Software
(as a Service)
Applications
Security
Databases
Operating Systems
Virtualization
Servers
Storage
Networking
Data Centers
Provider Managed

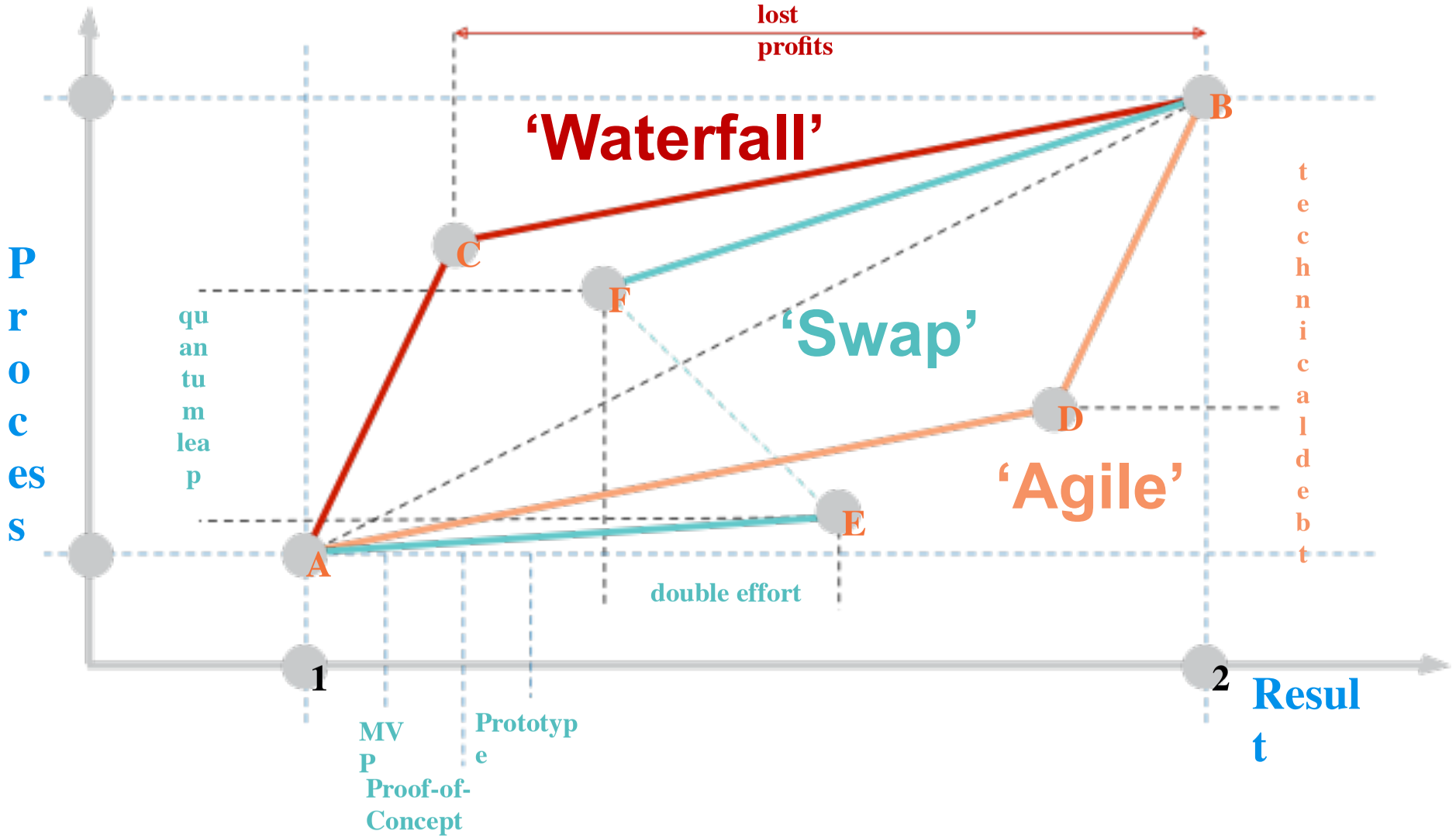
#noOPS #noPM #noDEV

- noOPS – **maturity** of the dev tool and frameworks are already raised at the level where every developer **can handle all/subset** of **Ops** activities
- noPM – communication & collaboration tools allow to build **peer-2-peer** channels and avoid bottlenecks
- noDEV – **Best Software is a Software was never written**. Huge number of open source and proprietary software shits development into integration side

Development Methodologies

Waterfall, Agile and more …

Result and Process, Result vs Process



S.M.A.R.T.

- ***Specific*** – target a specific area for improvement.
- ***Measurable*** – quantify or at least suggest an indicator of progress.
- ***Assignable*/Achievable** – specify who will do it.
- ***Realistic*** – state what results can realistically be achieved, given available resources.
- ***Time-related/Time-bound*** – specify when the result(s) can be achieved.

Software Architecture

UML, Patterns …

Topics

- Programming paradigms
- External software quality factors
- Design patterns & UML & Best practices
- Legacy systems & Technical debt & Refactoring
- Compilers & Interpreters & Metaprogramming
- Errors Handling
- Multi threading, Client-Server, Peer2Peer
- Orthogonal API

Structured programming, Imperative programming, Procedural programming

- Separated code & data scopes
- Global functions
- Assignment operator
- Execution: set of commands are changing global program state

Functional programming

- Lambda calculus
- Everything is a function
- Lists, trees
- No explicit assignment operator (explicit variables)
- No explicit program state

```
(defun factorial (n)
  (if (= n 0)
      1
      (* n (factorial (- n 1))) ) )

(loop for i from 0 to 16
  do (format t "~D! = ~D~%" i (factorial i)) )
```

Logic programming

- Facts & rules
- Theorems
- Mathematical Logic

```
man(Vlad).
woman(Tanya).
family(X, Y):-man(x),woman(Y).

?- family(Vlad, Y)
Y = Tanya
```

Object-oriented programming

- Encapsulation
- Polymorphism (overloading, virtual methods, static & dynamic type)
- Inheritance (multiple inheritance, interfaces, abstract classes)
- Aggregation & Composition
- Generic programming/Meta-programming
- Meta-classes

```
template <int N> struct Factorial {
  enum { value = N * Factorial<N - 1>::value };
};
template <> struct Factorial<0> {
  enum { value = 1 };
};
// Factorial<4>::value == 24
// Factorial<0>::value == 1
void foo() {
  int x = Factorial<4>::value; // == 24
  int y = Factorial<0>::value; // == 1
}
```

Context-free grammar

- Set of production rules that describe all possible strings in a given formal language

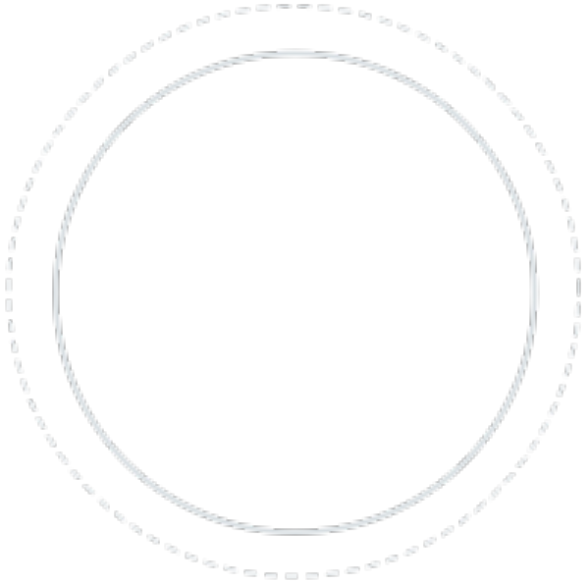
Digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Digits → Digit | Digit Digits
Number → Digits | (-Digits)
Exp → Number
Exp → Exp + Exp
Exp → Exp – Exp
Exp → Exp * Exp
Exp → Exp / Exp
Exp → (Exp)

Programming languages/software development evolution

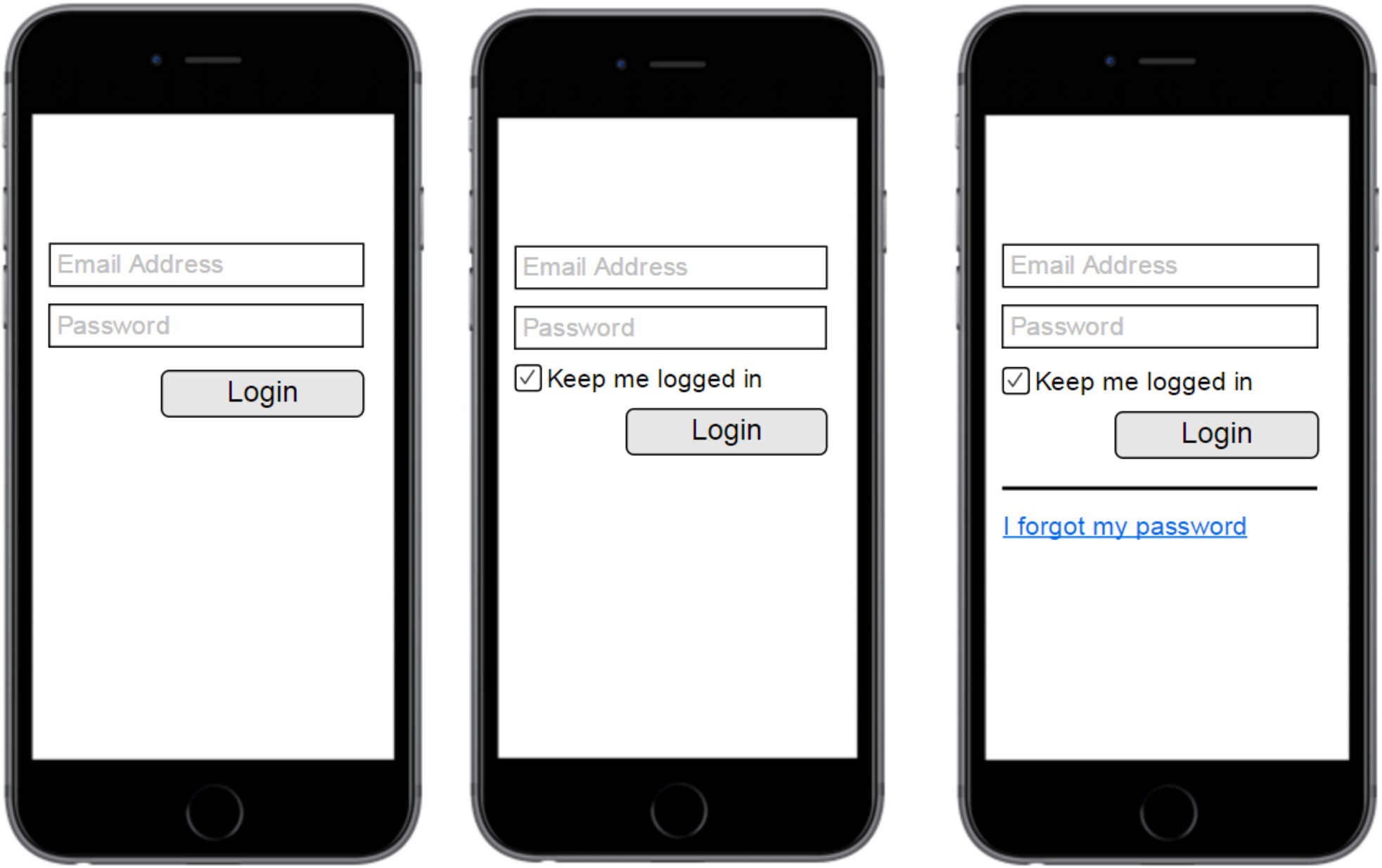
- Key paradigms will stay
- Distributed environments
- Functional trend
- Dynamic resource management
- Open source
- Libraries, Frameworks, Tools, IDEs
- Hybrid clouds
- ML & Big Data

- Изучая патологию, мы начинаем понимать норму

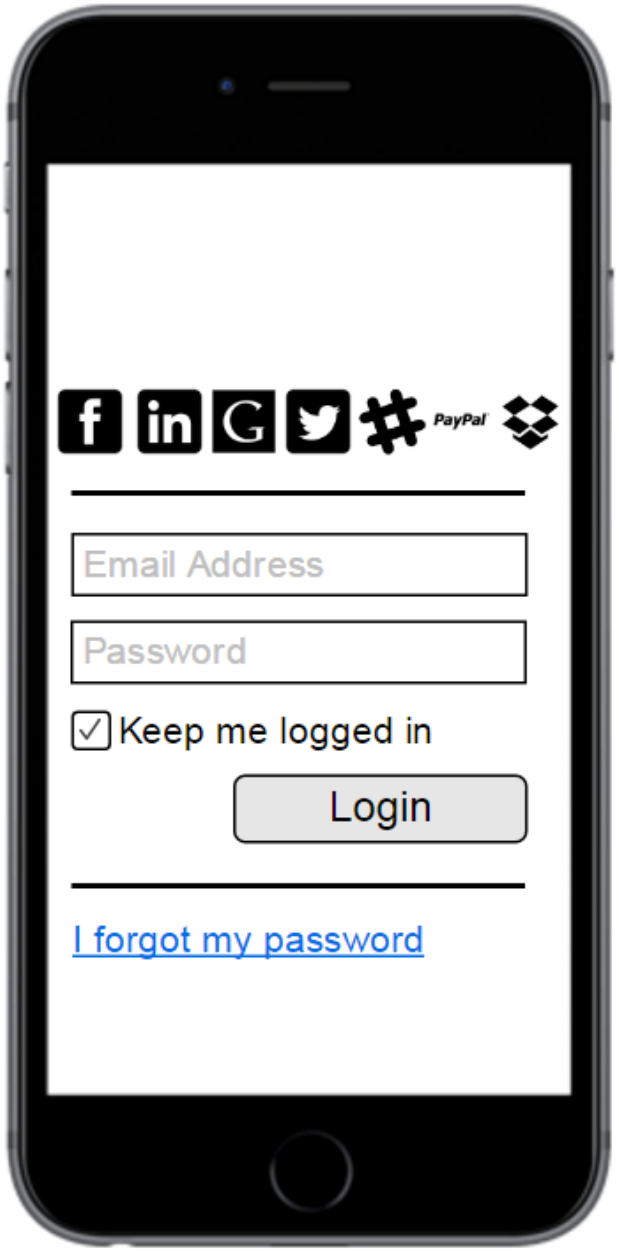
External software quality factors



Evolution


Email Address
Password
Login
Email Address
Password
Keep me logged in
Login
Email Address
Password
Keep me logged in
Login
I forgot my password

Evolution


PayPal
Email Address
Password
Keep me logged in
Login
I forgot my password


Continue

Evolution

```
void login(
   std::string const& email,
   std::string const& pass) {
}
```

```
class provider {};
class service: public provider {};
class facebook: public provider {};
class linkedin: public provider {};
// etc.
template<bool const T>
void login(T liveForever) {
}

template<class T, class... Args>
void login(T provider, Args... args) {
   // login with provider
   login(args...);
}
```

Key factors

- Correctness (Корректность)
- Robustness (Устойчивость)
- Extendibility (Расширяемость)
- Reusability (Повторное использование)

- Compatibility (Совместимость)
- Efficiency (Эффективность)
- Portability (Переносимость)
- Easy of Use (Простота использования)
- Functionality (Функциональность)
- Timeliness (Своевременность)
- Verifiability (Верифицируемость)
- Integrity (Целостность)
- Repairability (Восстанавливаемость)
- Economy (Экономичность)

Design review check list

Correctness (Корректность)

- Do we cover all features from specification?
- Do we define clear borders between our system and external elements?

Robustness (Устойчивость)

- Do we list all possible errors/emergency situations?
- Do we have clear plan how to handle, fix and restore after all this issues?

Design review check list

Extendibility (Расширяемость)

- Do we define which patterns and where we will implement?
- Do we plan a few design iterations?
- Do we have resources to pay technical debt?
- What is level of coupling for our system?

Reusability (Повторное использование)

- Do we use maximum number of open-source & proprietary ready-2-use solutions?
- Best code is code was not written.
- Do we have clear vision how and where we will reuse our components?

Design review check list

Compatibility (Совместимость)

- Do we use open/well-define/industry-proved standards?
- Do we follow SOLID principles?
- Patterns, again.

Efficiency (Эффективность)

- Do we define acceptable resonance time for all our communication channels?
- Do we define minimal hardware/software requirements?

Design review check list

Portability (Переносимость)

- Are we cross-platform?
- How may OS should we support, versions?
- Do we have DB abstraction layer?
- Do we design orthogonal APIs?

Easy of Use (Простота использования)

- Who are our users?
- How much time do we need to setup dev/qa/prod environments?
- Do we have auto generated source code documentation?
- What is size of user manual?

Design review check list

Functionality (Функциональность)

- Do we understand 80%/20% principle?
- Do we have prioritized back-log for current release? For one year from now?

Timeliness (Своевременность)

- Do we know our competitors?
- Do we have clear vision where domain goes?

Design review check list

Verifiability (Верифицируемость

- Do we plan to use TDD – Test Driven Development?
- Integration tests?
- Automated UI tests?
- HA tests?
- Logging & Audit

Integrity (Целостность)

- Security?

Design review check list

Repairability (Восстанавливаемость)

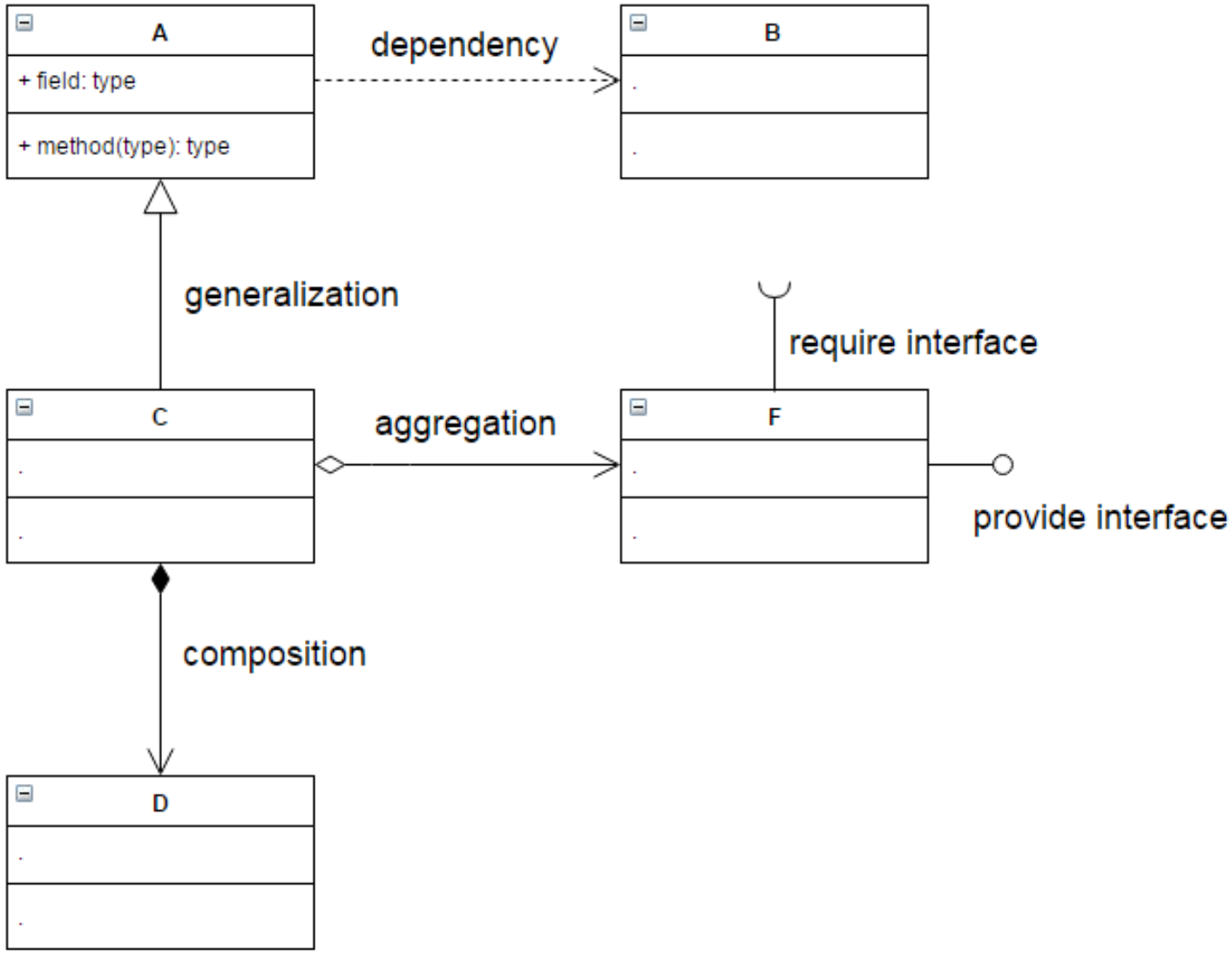
- Monitoring?

Economy (Экономичность)

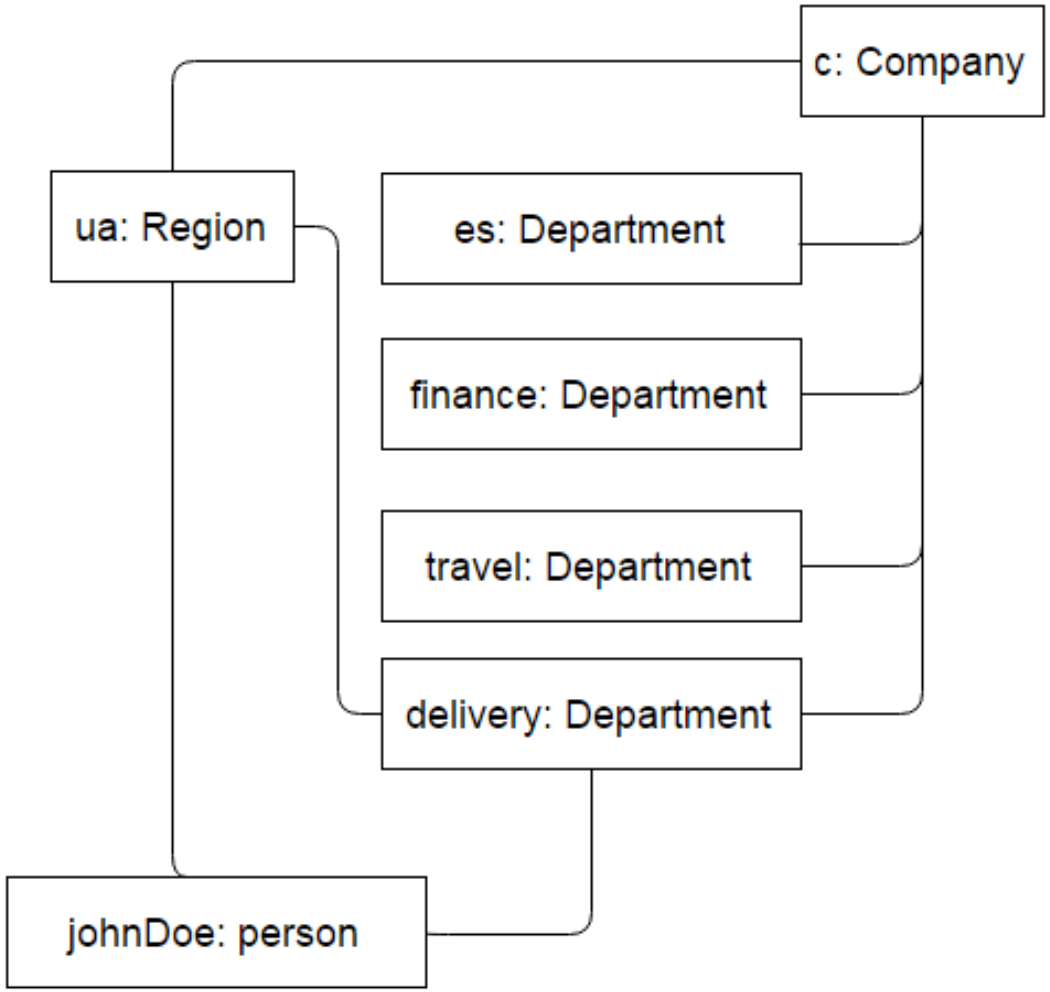
- Estimates?

UML & Design patterns, Best practices

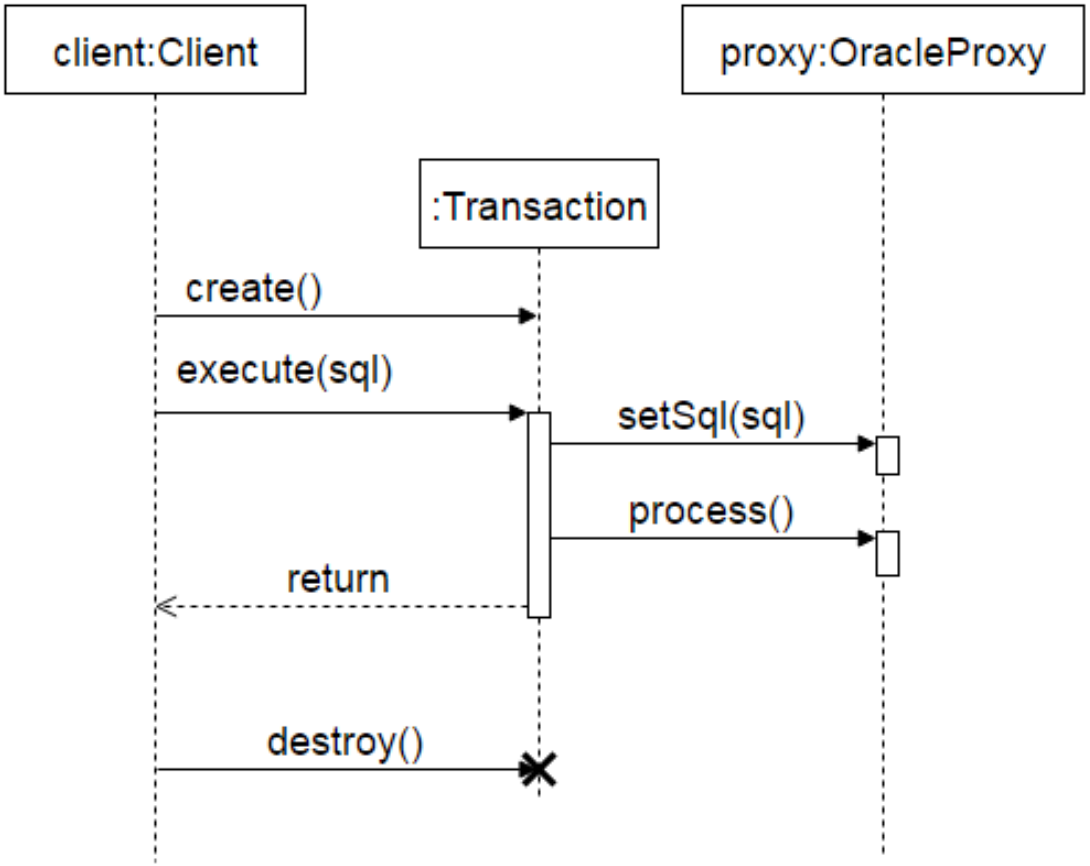
UML - Classes diagrams



UML - Objects diagrams



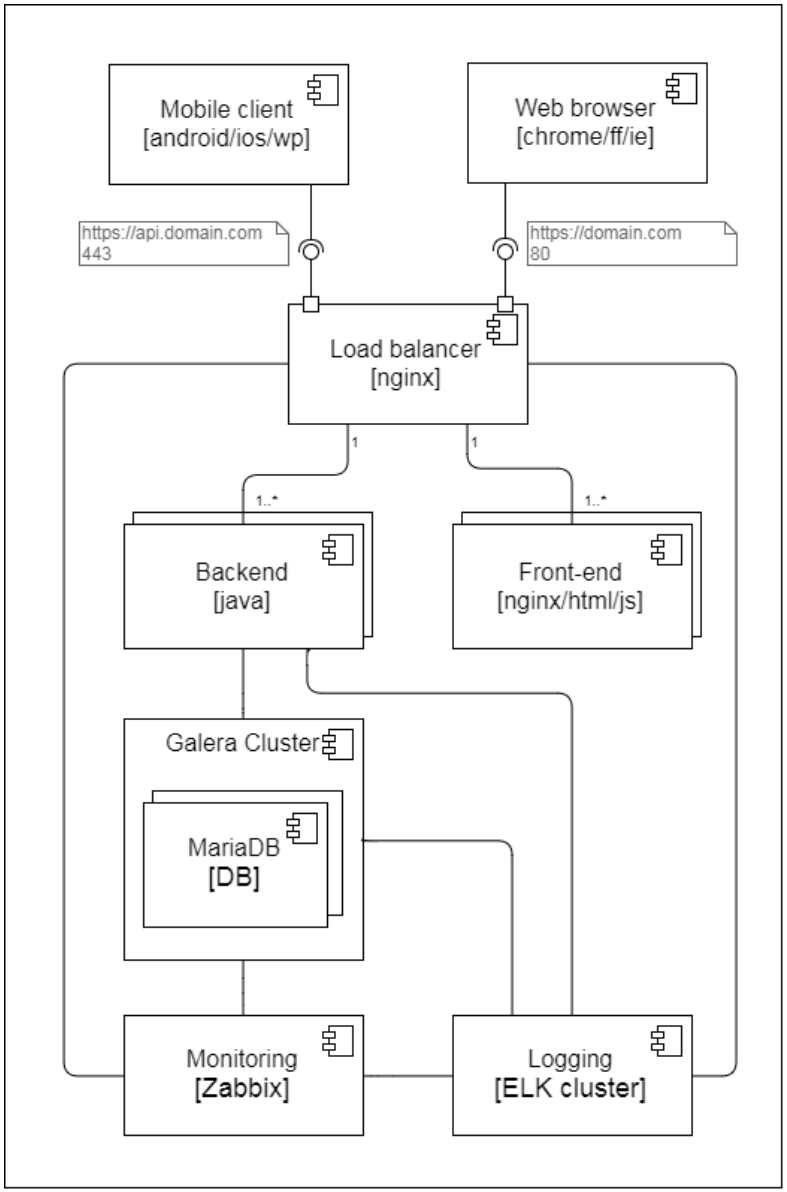
UML - Sequence Diagrams



Dropbox-like service: high-level requirements

- Automatic file sub-system synchronization between single backend server and multiple consumer applications
- Version based back API
- Web client
- Standalone clients Win/Mac/Linux
- Mobile client
- Full Audit subsystem
- Versioning subsystem for stored files

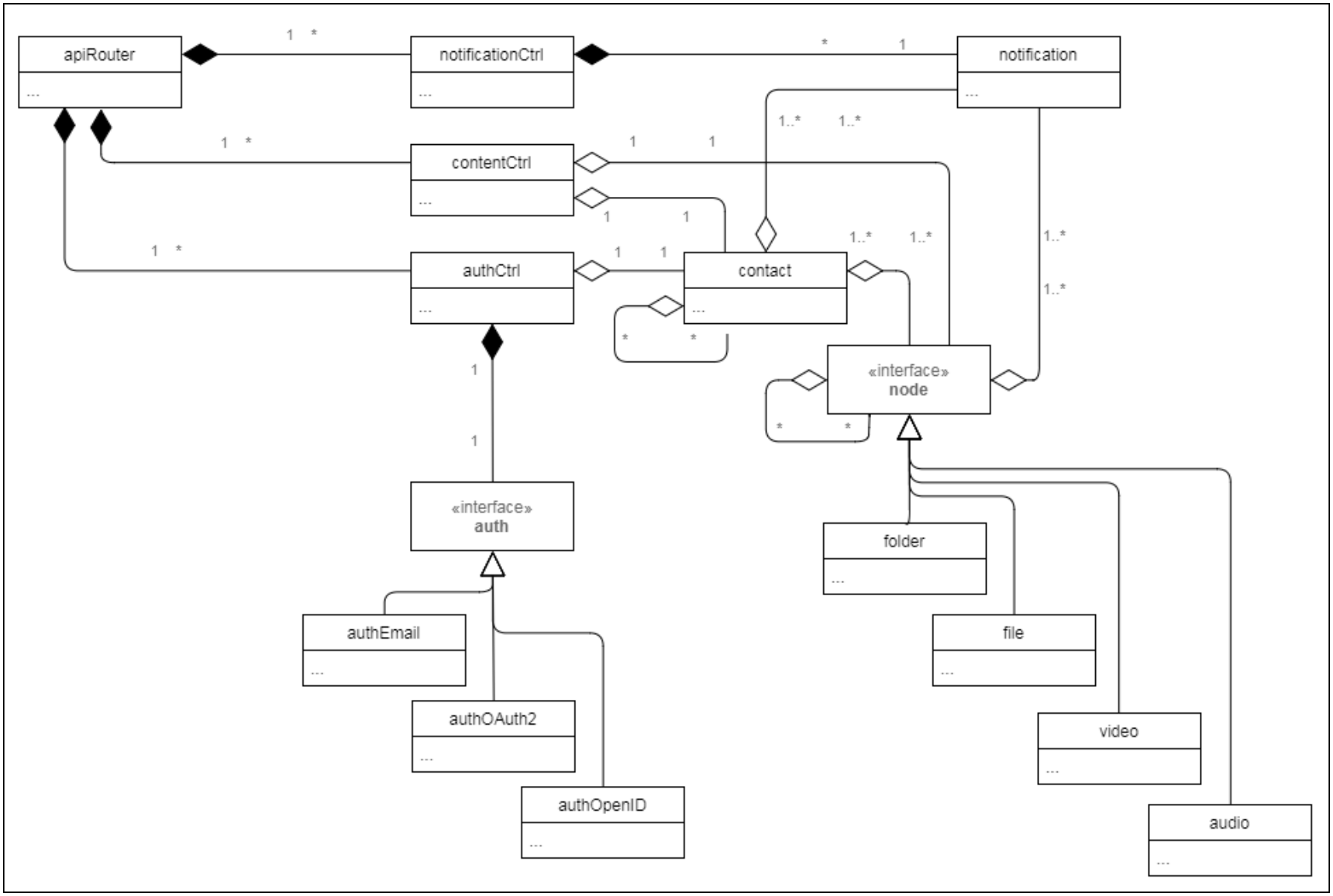
Dropbox-like service: high-level requirements



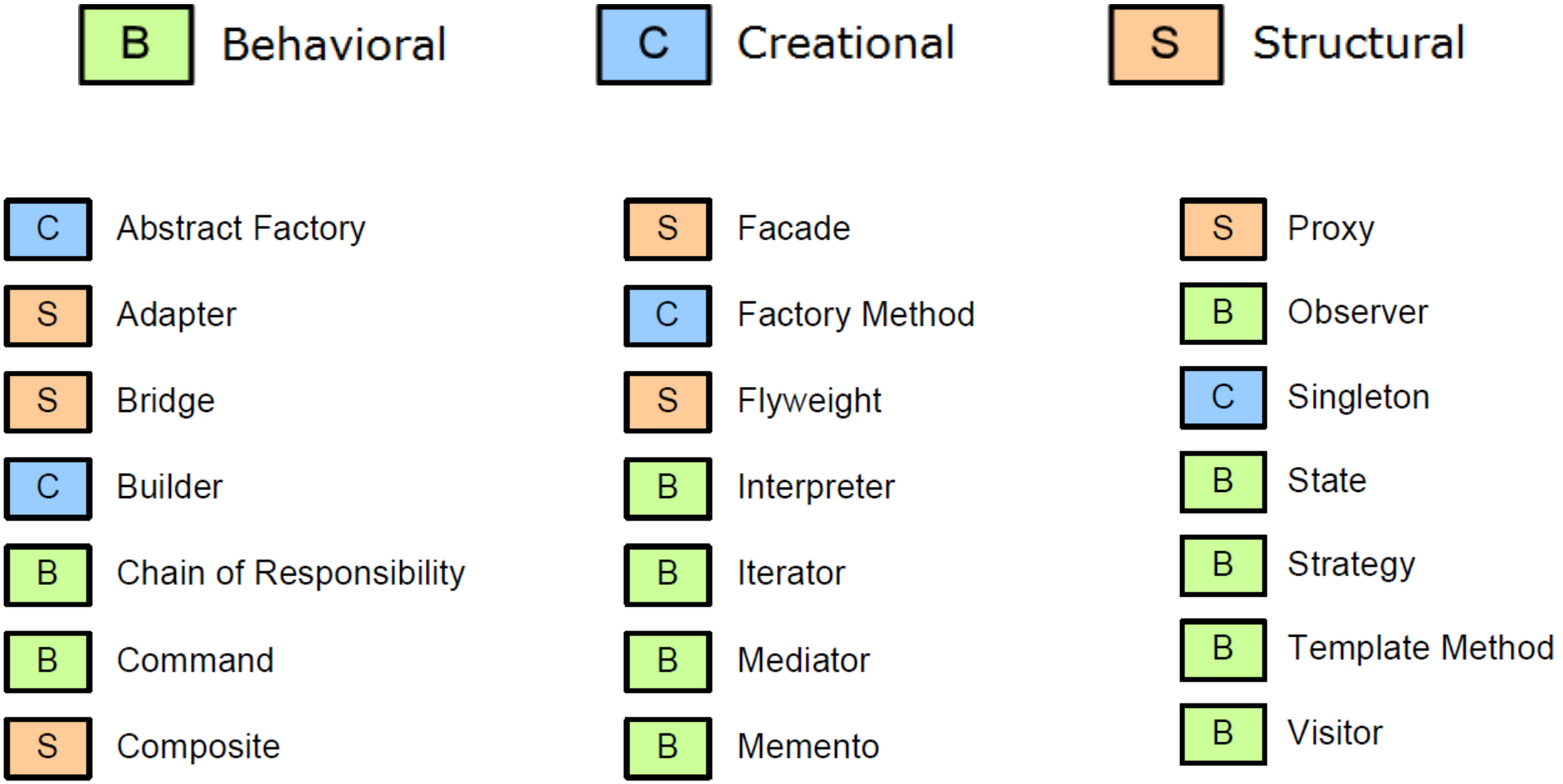
Dropbox-like service: user management

- Registration/login via email
- Registration/login using OAuth2
- RBAC – Role Based Access Control
- Hierarchy group management
- RESTful API, versioning
- Notification subsystem
- Blocked users list

Dropbox-like service: user management



Design patterns

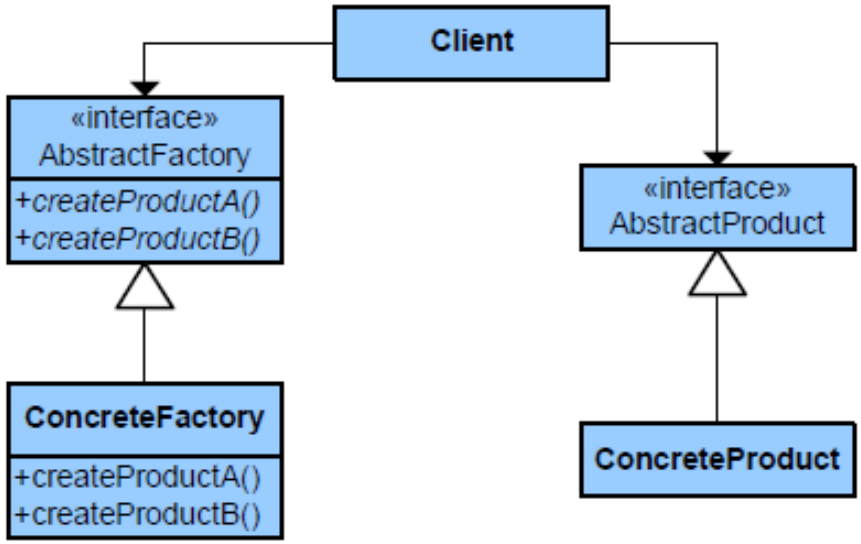


Design patterns

Abstract Factory

Type: Creational

What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.

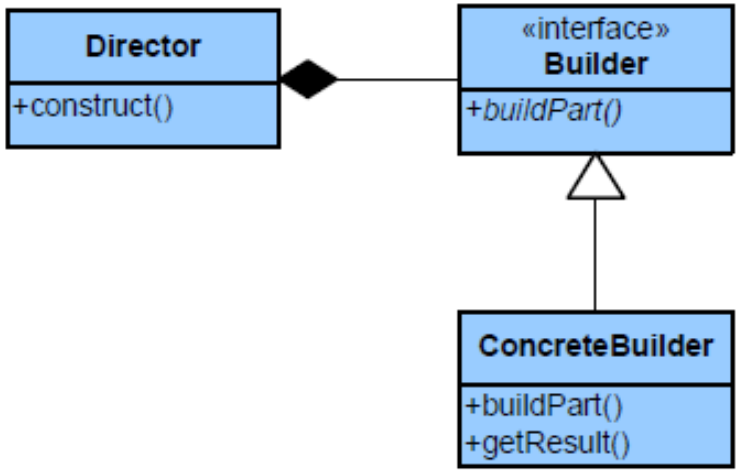


Xwindows/QT

Builder

Type: Creational

What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.



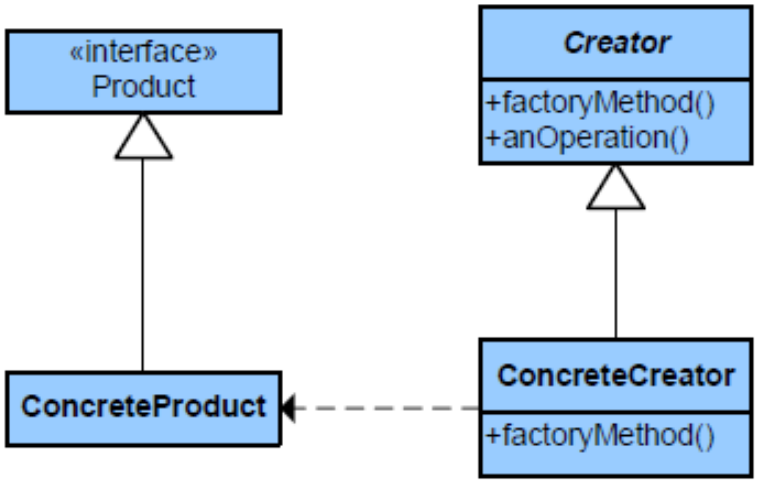
Load document from different formats

Design patterns

Factory Method

Type: Creational

What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.

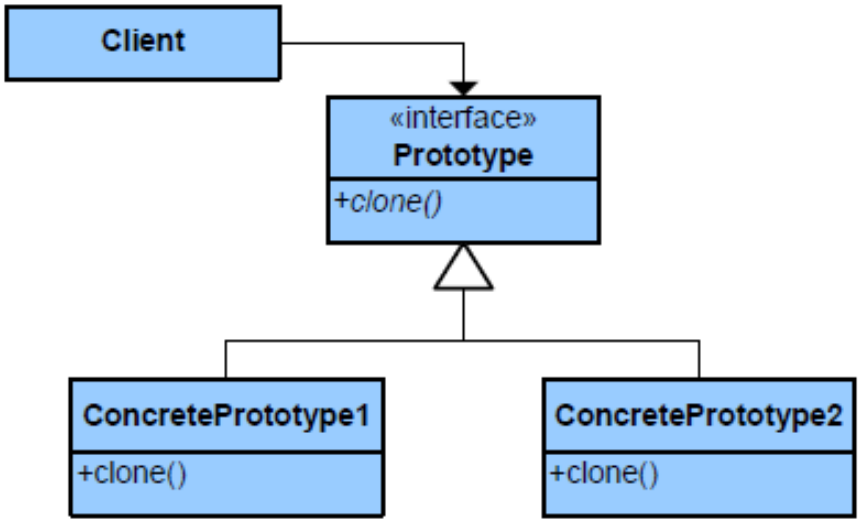


Virtual constructor.

Prototype

Type: Creational

What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.



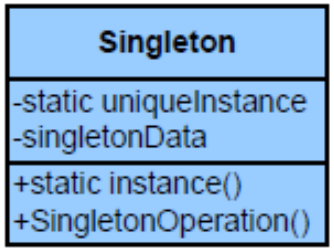
"heavy" object, when copy is much quickly than creation from scratch, reuse some parts

Design patterns

Singleton

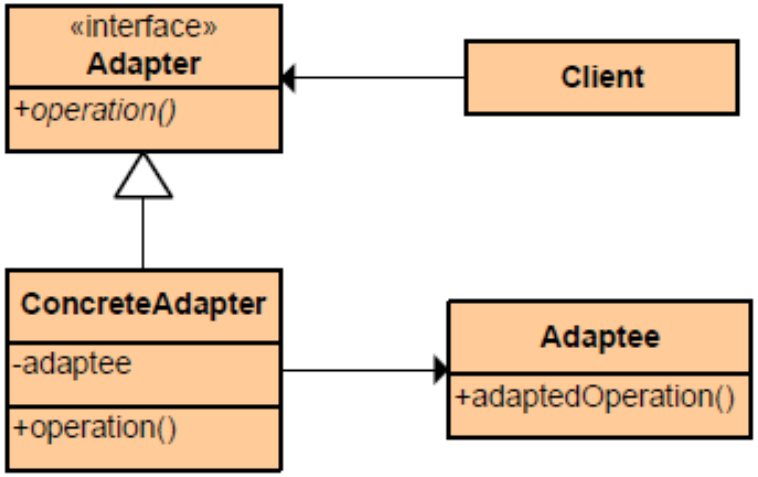
Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.



Logger

Design patterns

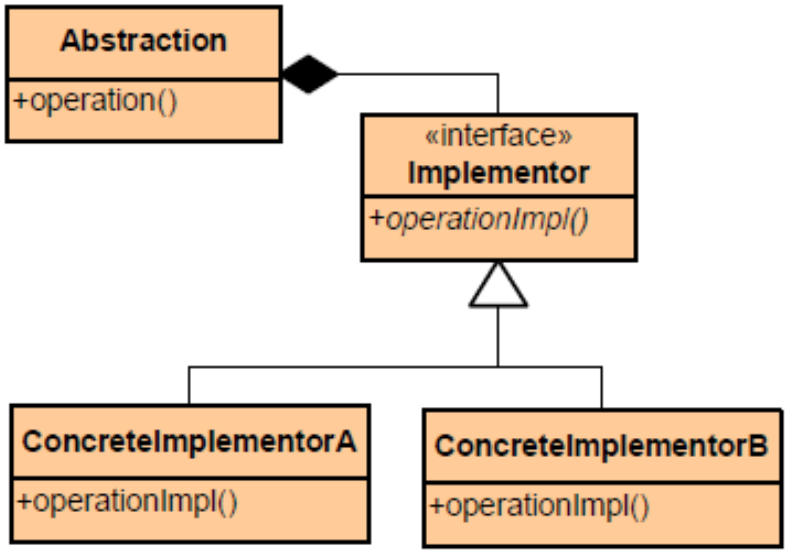


Adapter

Type: Structural

What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

Use XPath to navigate JSON



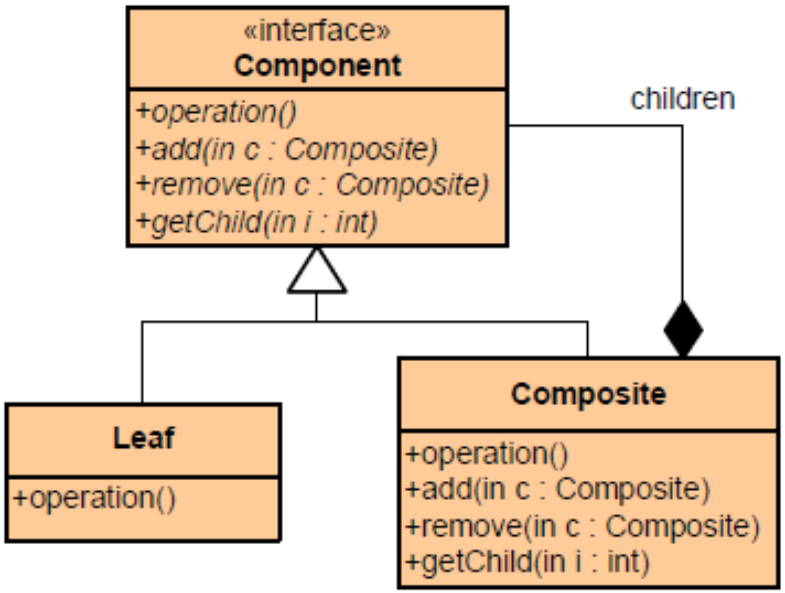
Bridge

Type: Structural

What it is:
Decouple an abstraction from its implementation so that the two can vary independently.

PIMPL – Point to Implementation

Design patterns

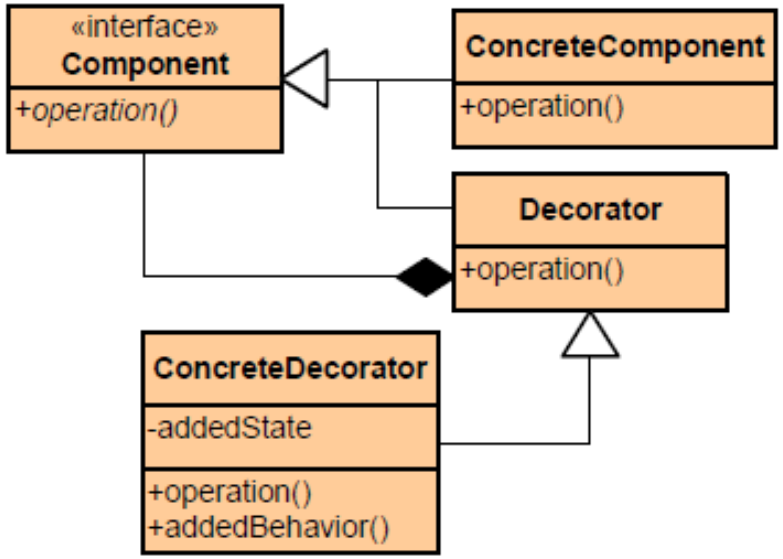


Composite

Type: Structural

What it is:
Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions of objects uniformly.

GUI windows, Multi-selection



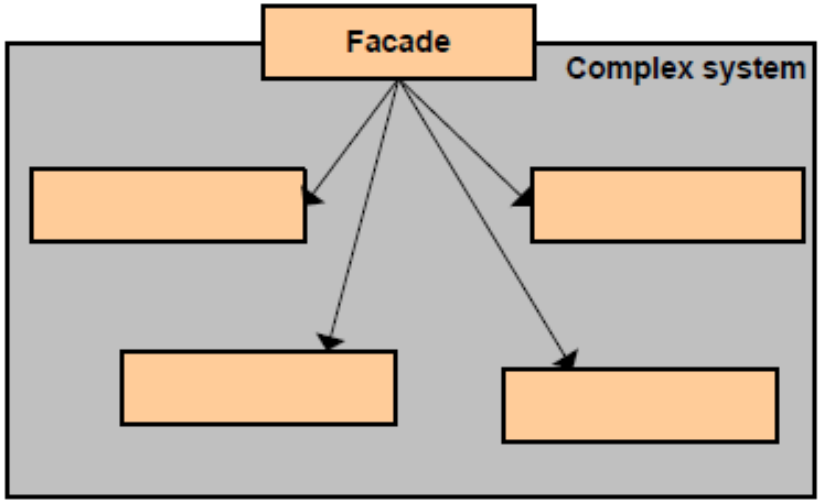
Decorator

Type: Structural

What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.

Serialization algorithms

Design patterns

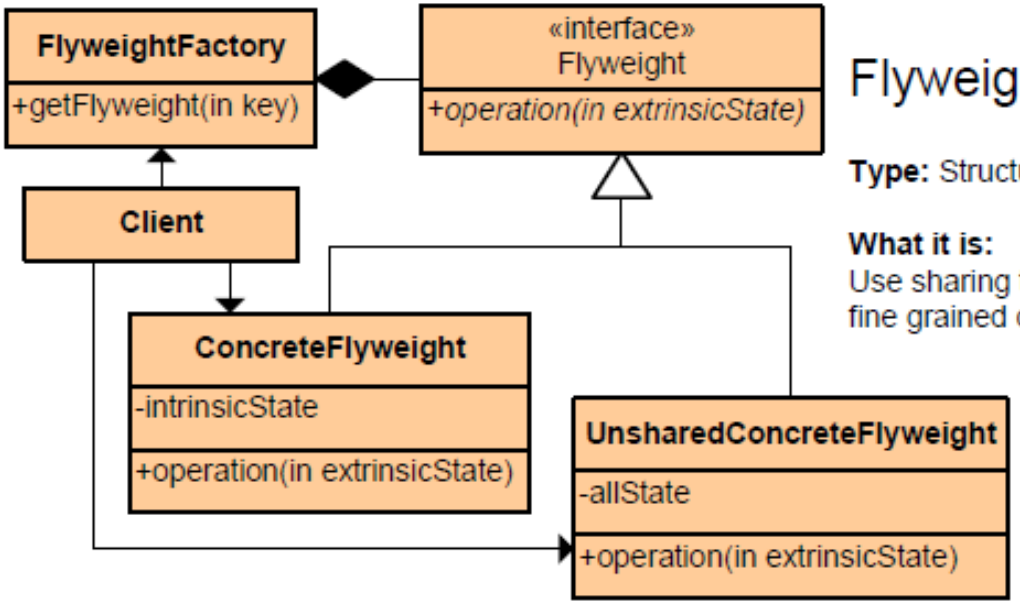


Facade

Type: Structural

What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

One-click-buy-button



Flyweight

Type: Structural

What it is:
Use sharing to support large numbers of fine grained objects efficiently.

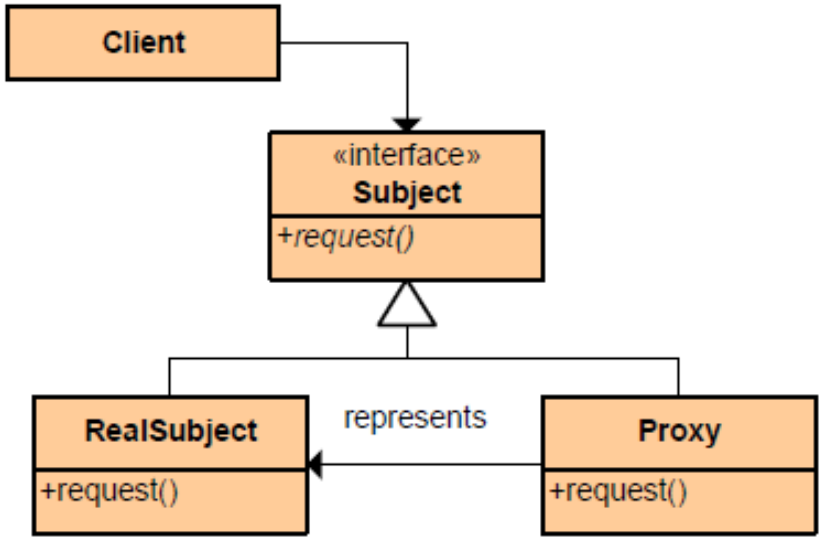
?

Design patterns

Proxy

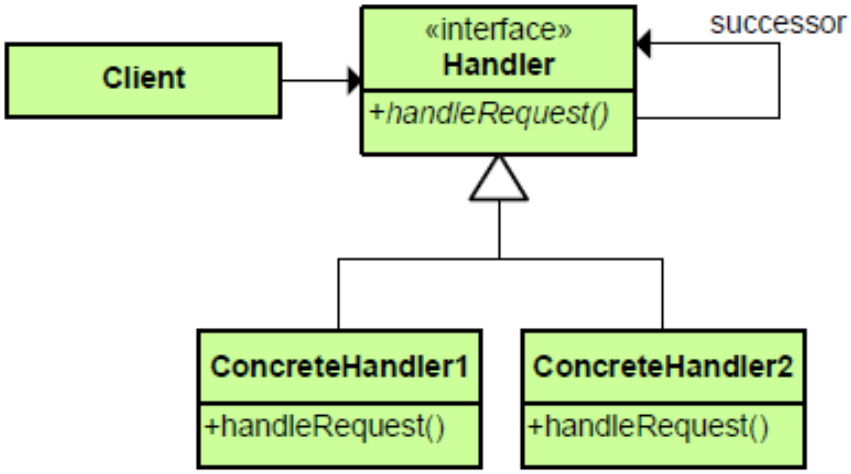
Type: Structural

What it is:
Provide a surrogate or placeholder for another object to control access to it.



Mocks, fakes

Design patterns

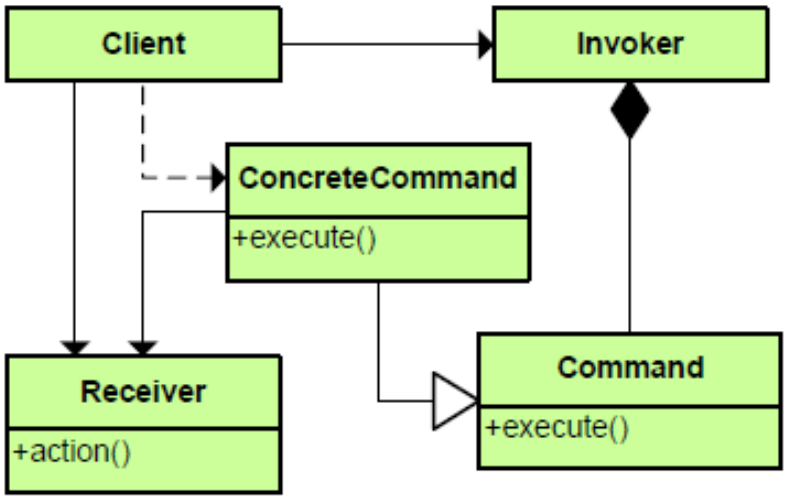


Chain of Responsibility

Type: Behavioral

What it is:
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Event handler in browser



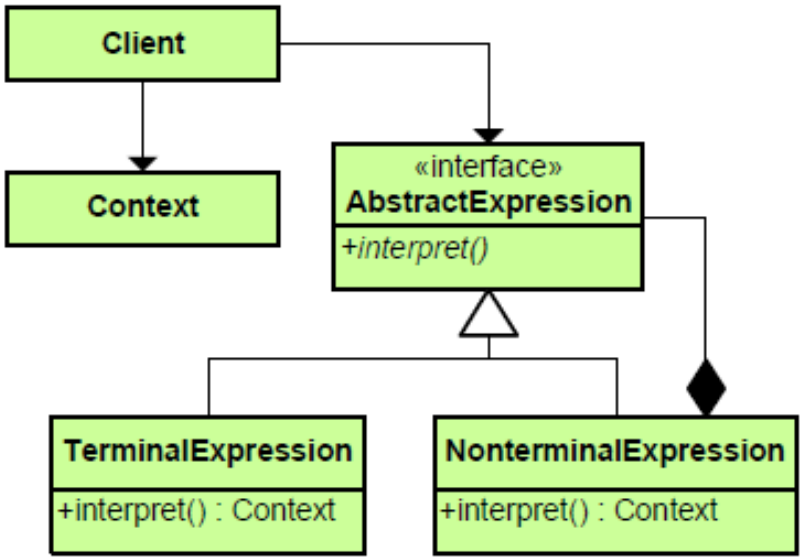
Command

Type: Behavioral

What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Undo/Redo

Design patterns

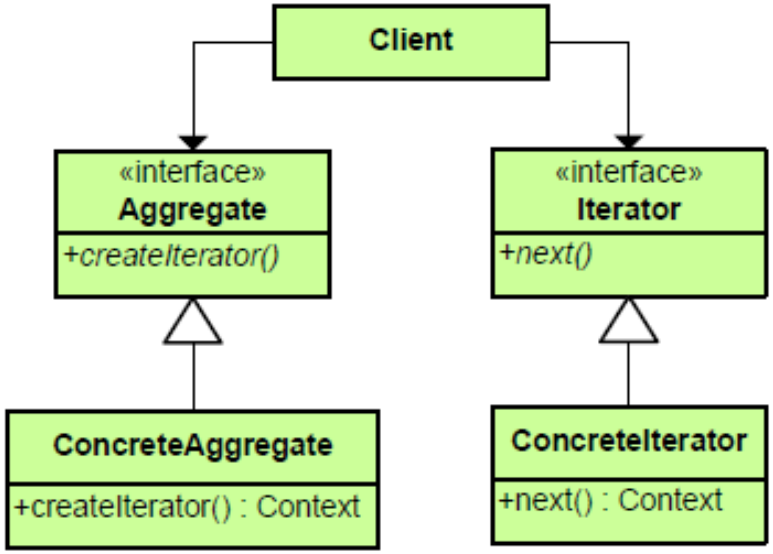


Interpreter

Type: Behavioral

What it is:
Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

DSL



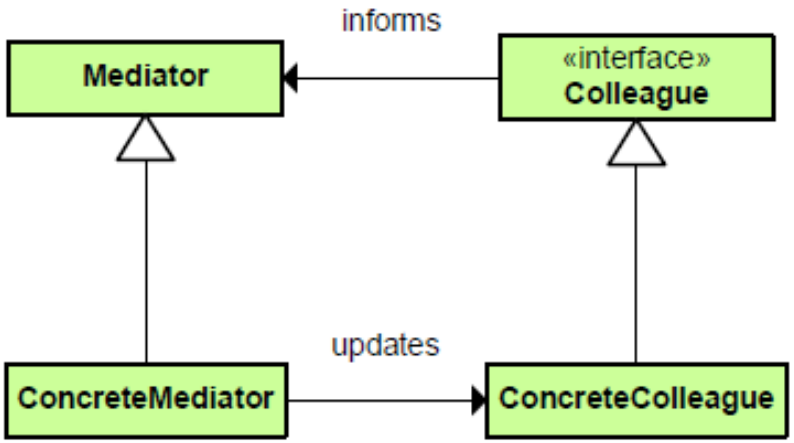
Iterator

Type: Behavioral

What it is:
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

foreach, pairs, intervals

Design patterns



Mediator

Type: Behavioral

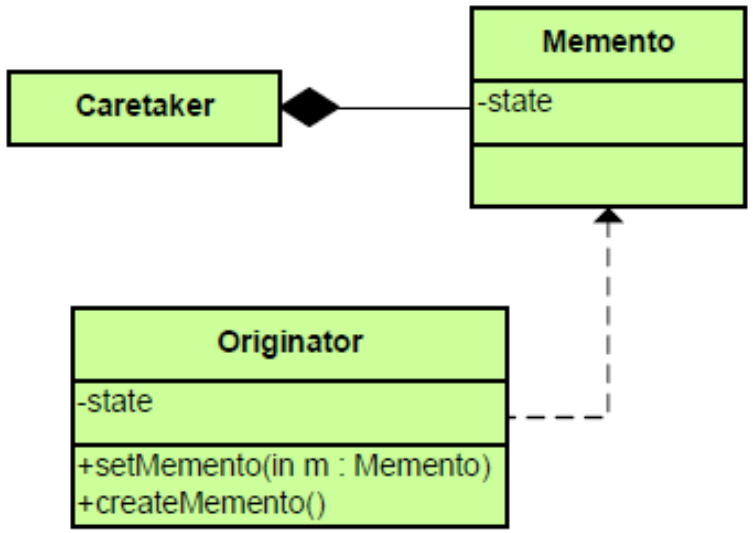
What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

MVC

Memento

Type: Behavioral

What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.



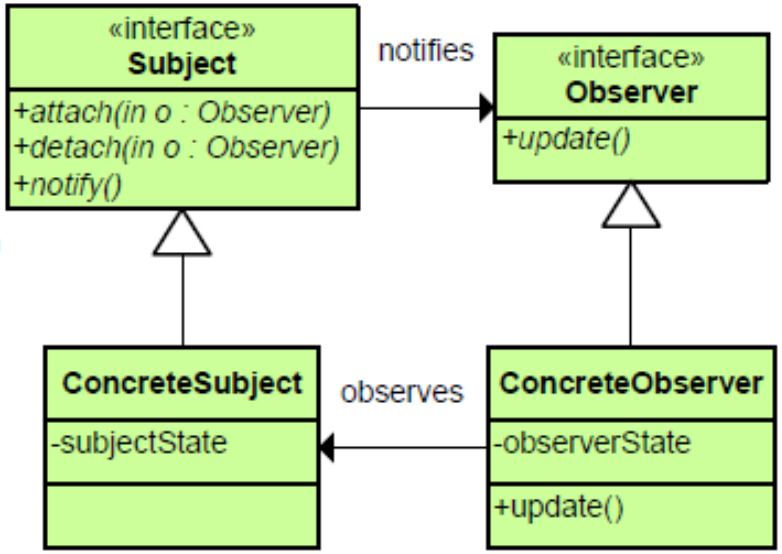
Save/Load
Serialization

Design patterns

Observer

Type: Behavioral

What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

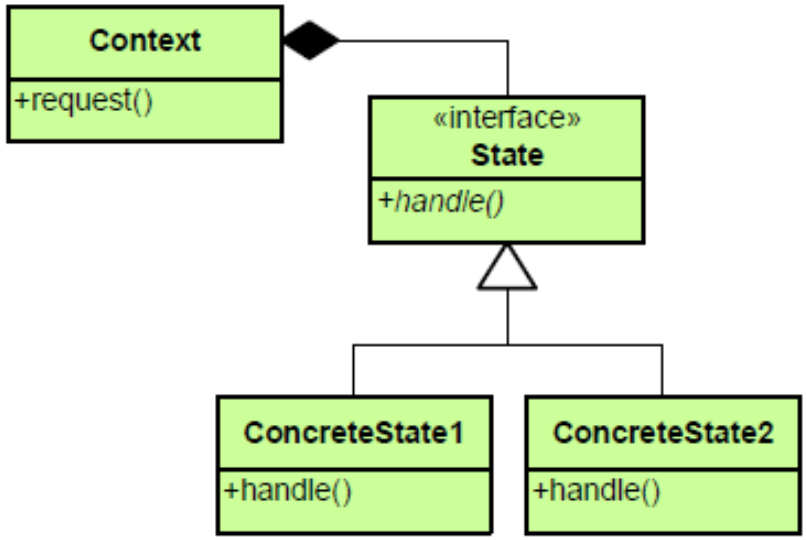


Broadcast mechanism

State

Type: Behavioral

What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.



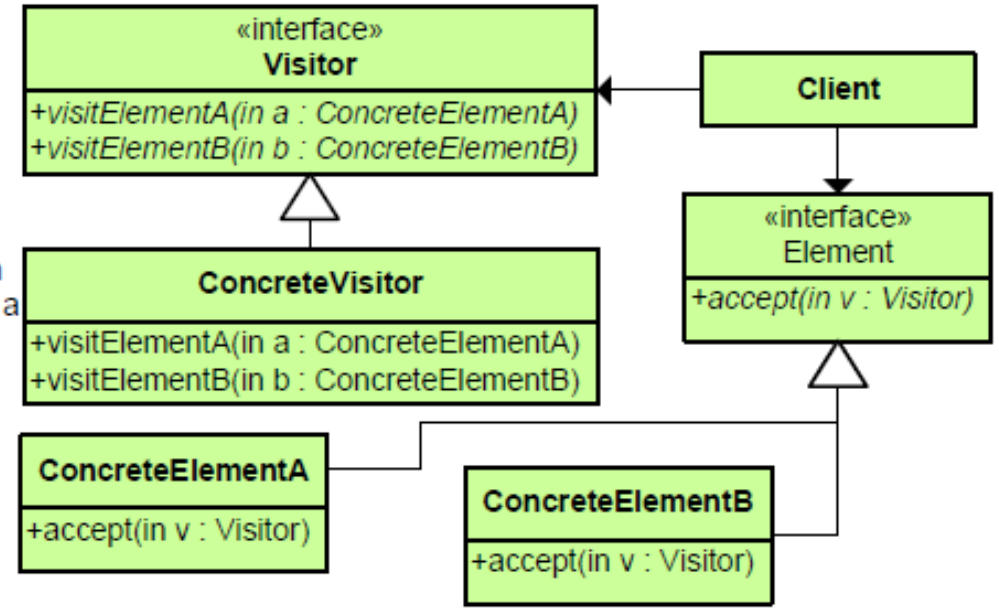
Finite State Machine

Design patterns

Visitor

Type: Behavioral

What it is:
Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.



Render hierarchy of object using different toolset (GPU/Sodt)

Law of Demeter

- Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.
- Each unit should only talk to its friends; don't talk to strangers.
- Only talk to your immediate friends.

Best practices

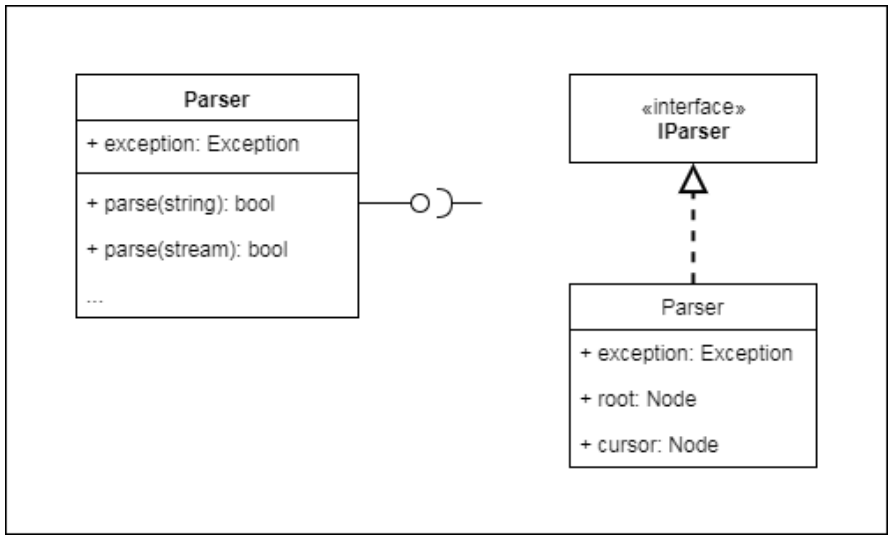
- PP Pareto Principle
- KISS Keep It Short and Simple
- YAGNI You Aren't Gonna Need It
- NIH Not Invented Here
- HIN Habit Inhibiting Novelty
- DRY Don't Repeat Yourself
- OAOO Once And Only Once
- SSOT Single Source Of Truth
- SVOT Single Version Of Truth
- SoC Separation of Concerns
- YOLO You Only Load it Once
- POGE Principle Of Good Enough
- POLA Principle Of Least Astonishment

S.O.L.I.D.

- S - Single Responsibility Principle
- O - Open-Closed Principle
- L - Liskov Substitution Principle
- I - Interface Segregation Principle
- D - Dependency Inversion Principle

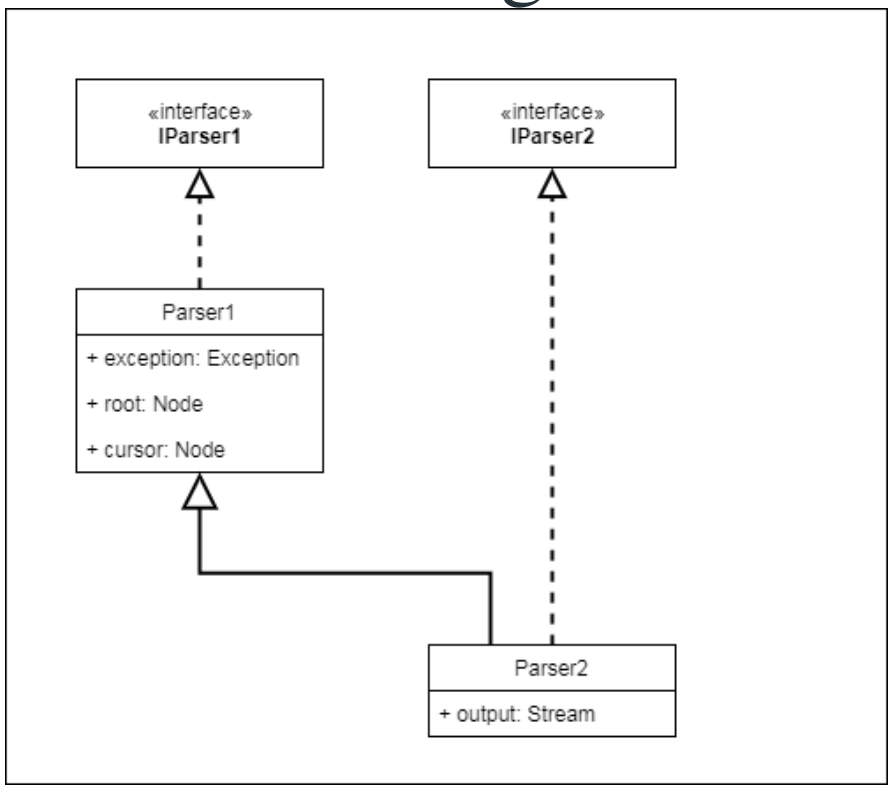
S.O.L.I.D. - Single Responsibility Principle

- **Single** interface to implement
- **Single** reason to modify



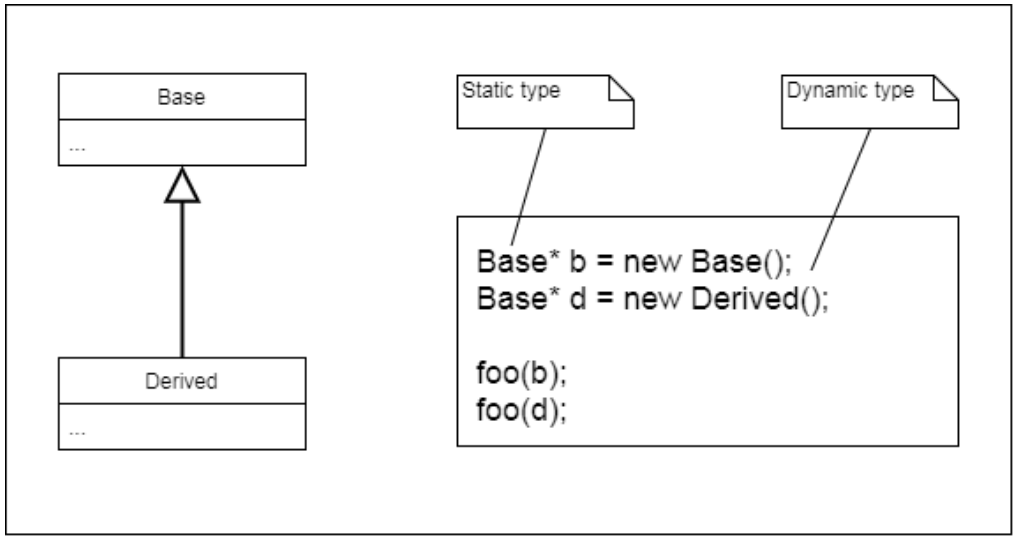
S.O.L.I.D. - Open-Closed Principle

- Interface should be **closed** for modification
- Interface can be **opened** for extension
- Interface is **opened** for errors fixing



S.O.L.I.D. - Liskov Substitution Principle

- **Derived** class instance **have** to be able to **use** anywhere **instead** of Base
- **Derived** class can make **pre conditions weaker** and/or **post conditions stronger** (requiring less and guaranteeing more)

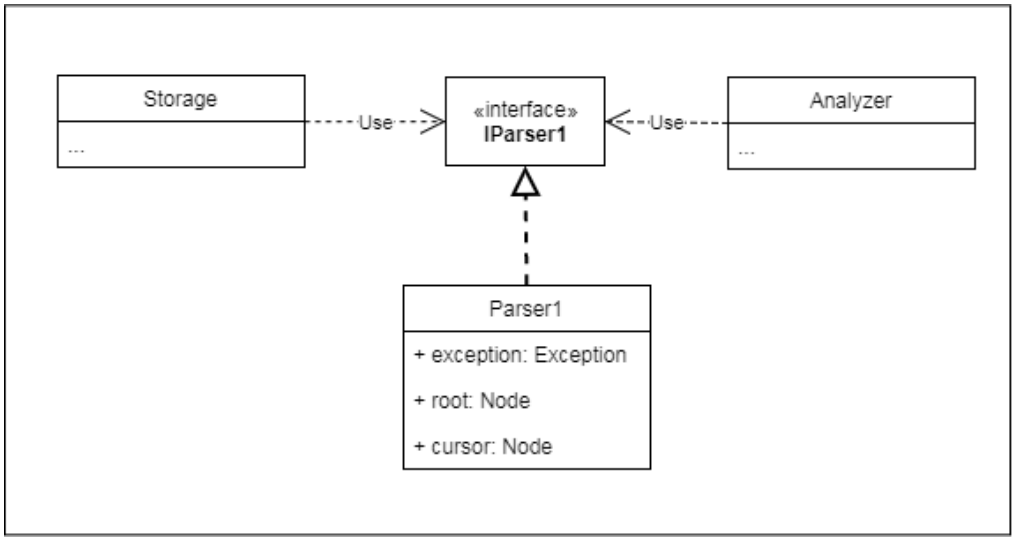


S.O.L.**I**.D. - Interface Segregation Principle

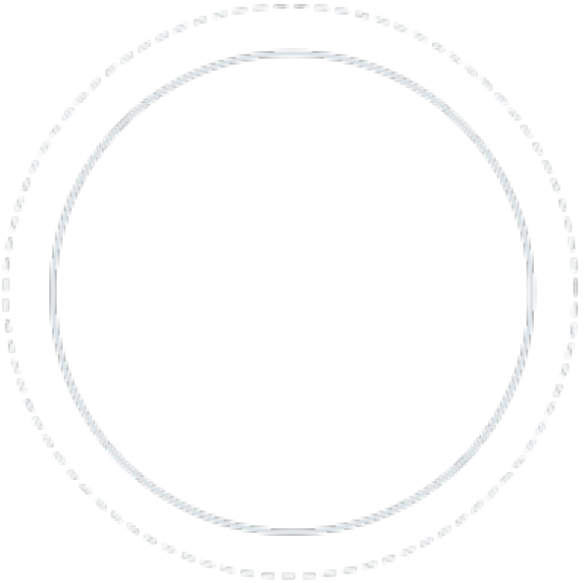
- Orthogonal interface

S.O.L.I.**D**. - Dependency Inversion Principle

- **Minimal relations** between classes
- Depends on **interface** not **implementation**



Legacy systems & Technical debt & Refactoring



Legacy system - How is this happening

Legacy system is a system where technical expertise/knowledge was lost

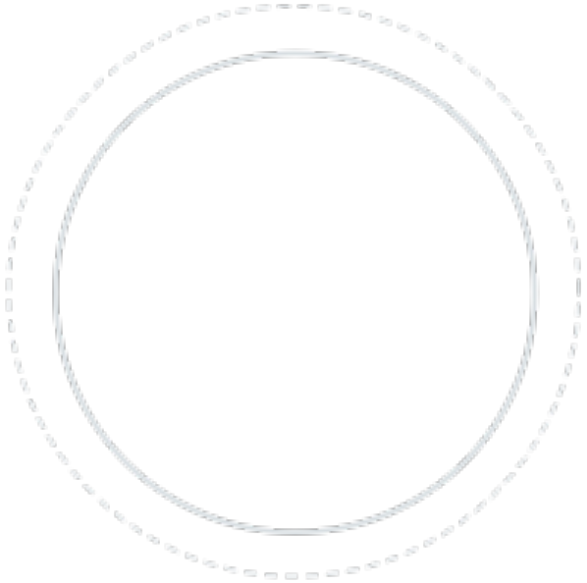
- Software with a history (~5+ years)
- A set of major releases
- Long evolution of source code
- Extending with new features
- Reuse inside new domains
- Integration with other systems

Legacy system is your code after one month without unit-tests and documentation

Refactoring

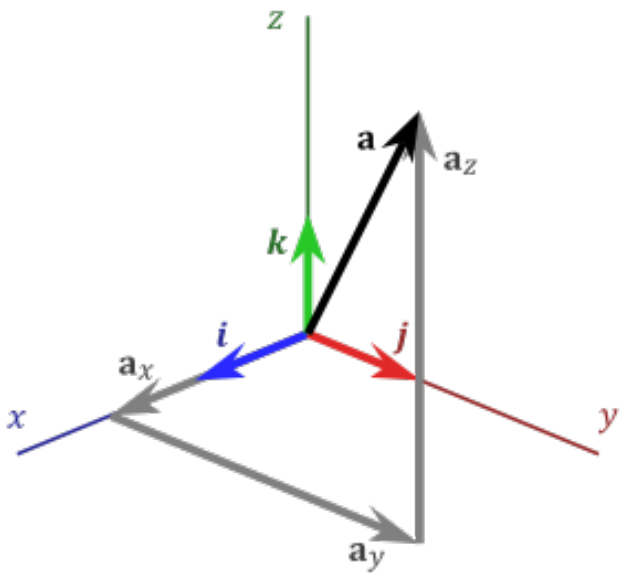
Increasing internal software quality without changing users' behavior (system interface)

- Decrease code coupling
- Clarify interfaces
- Remove God-objects
- Rewrite code using patterns
- Improve documentation
- e.t.c



Orthogonal API

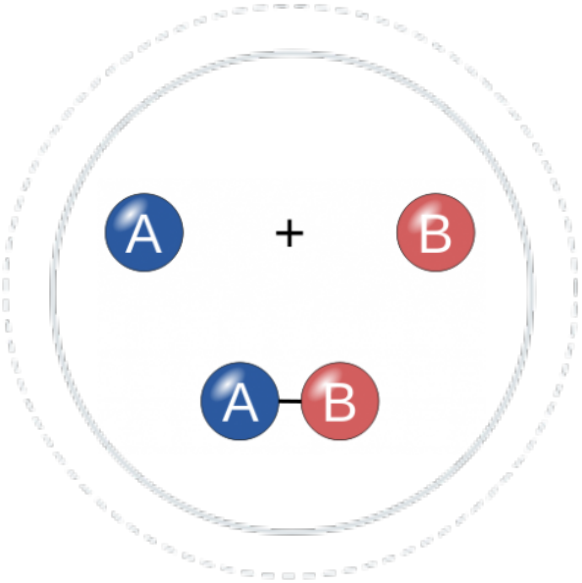
Orthogonal API




```
FILE * fopen ( const char * filename, const char * mode );
```

```
HANDLE WINAPI CreateFile(
  _In_ LPCTSTR lpFileName,
  _In_ DWORD dwDesiredAccess,
  _In_ DWORD dwShareMode,
  _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  _In_ DWORD dwCreationDisposition,
  _In_ DWORD dwFlagsAndAttributes,
  _In_opt_ HANDLE hTemplateFile
);
```

Minimal number of endpoints (functions, methods, RESTful end-points) which allows to have maximum coverage for target domain model.

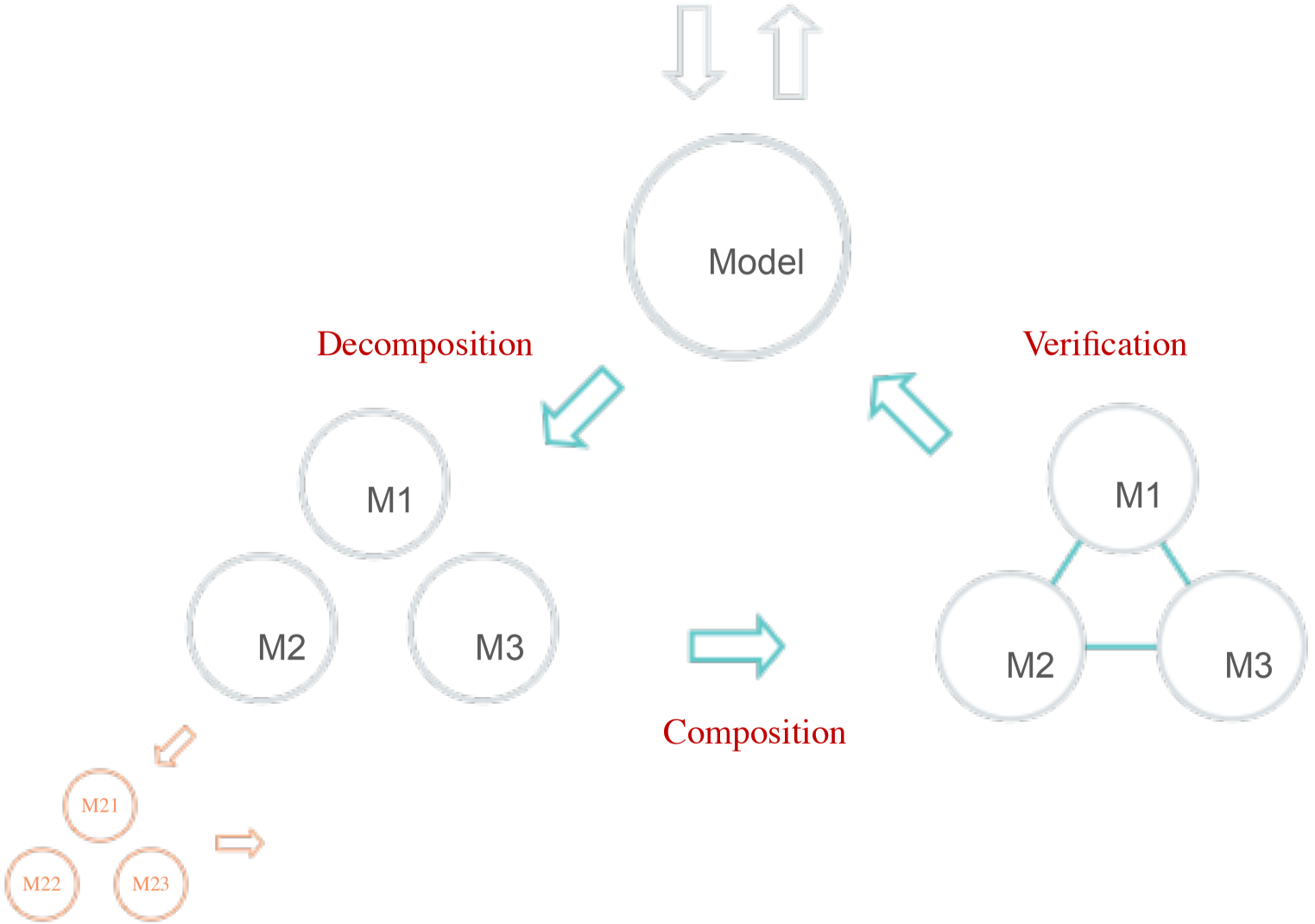


Decomposition
Composition
Verification

Why is Software Development complicated?

- Speed of tech changes, legacy products
- Misunderstanding between business owner, management and tech staff
- Flexibility of tech tools
- Math model will never be equal the original system

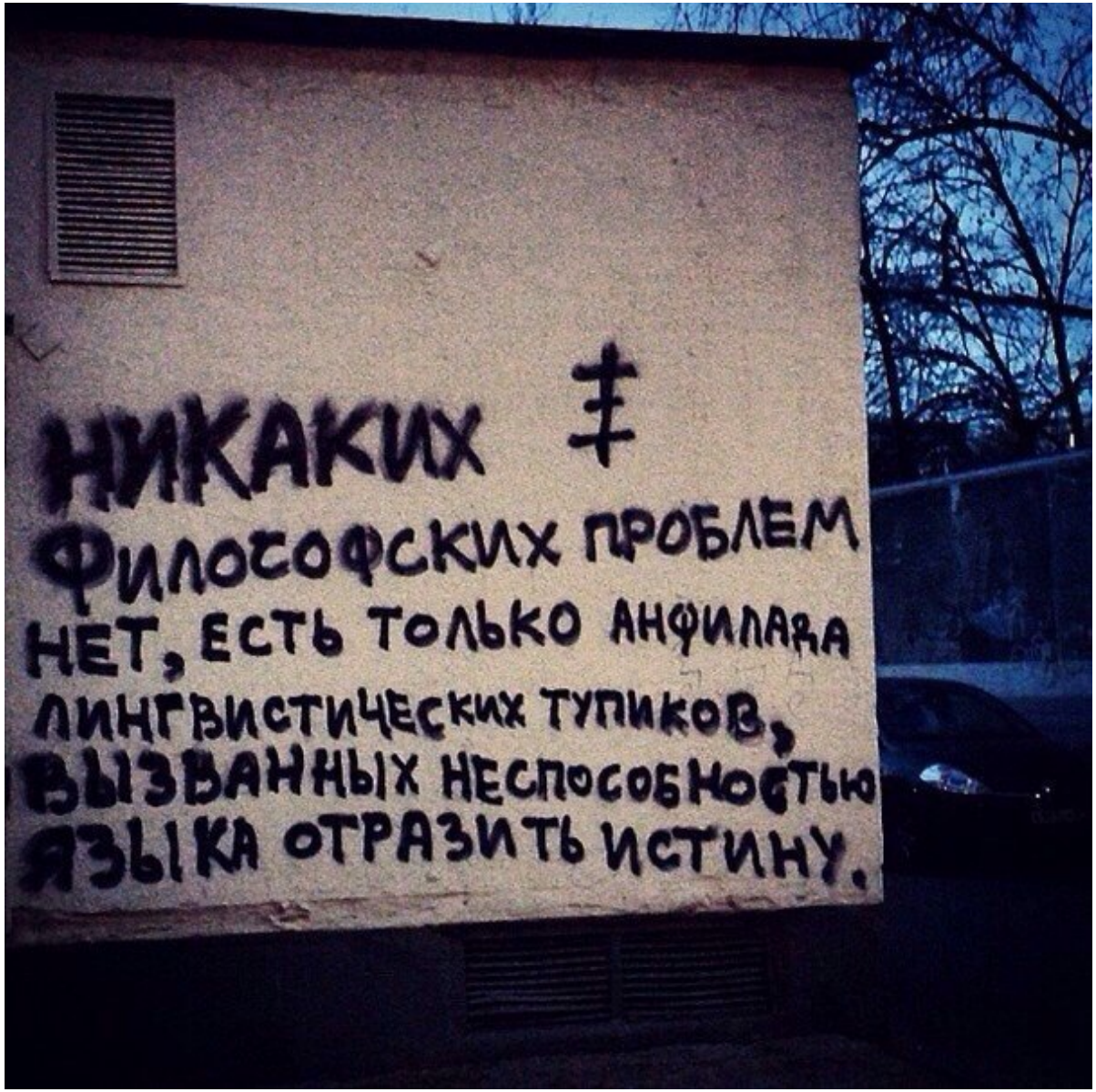
Cycle of life



Key Human been issue

Key Software Development issues

- Variable name
- Cache invalidation
- Number of abstraction layers



Errors Handling

Axioms

- Software always contains errors
- TDD & Testing couldn’t find all errors
- External environment changes couldn’t be controlled
- Software evolves

Error codes

- Use POD (plain old data) to define status of operation (integer, boolean, etc.)
- Sometimes mixing error status and result, 0 – error, non-zero result - success
- Specific variable as last operation status (allocated inside TLS – Thread Local Storage)
- RESTful API operation results

Error codes

```
std::string userName('');
if (getUser(userName)) {
// process user
} else {
// error, user not found
}
```

```
HANDLE hFile;
hFile = CreateFile(argv[1],        // name of the write
   GENERIC_WRITE,        // open for writing
   0,           // do not share
   NULL,           // default security
   CREATE_NEW,        // create new file only
   FILE_ATTRIBUTE_NORMAL,    // normal file
   NULL);           // no attr. Template

if (hFile == INVALID_HANDLE_VALUE) {
    DisplayError(TEXT("CreateFile"));
   _tprintf(TEXT("Terminal failure: Unable to open file \"%s\" for write.\n"), argv[1]);
   return;
}
```

Exceptions

- Hierarchy of classes defines a set of possible errors in the system
- Stack unwinding
- Additional code, generated by build system, to support exception handling, generating & catching

Exceptions

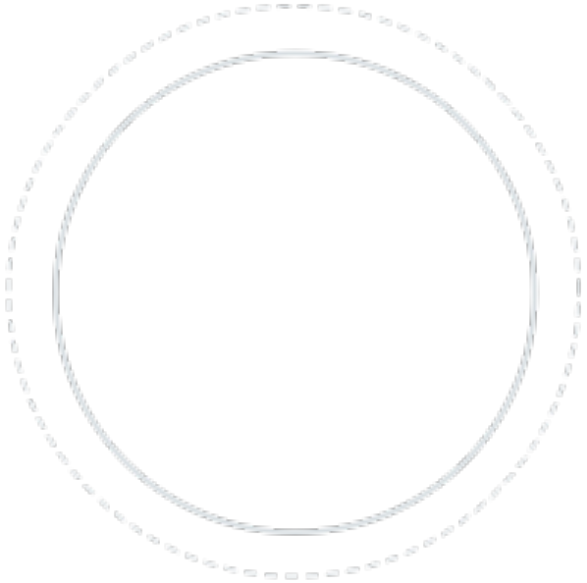
```
class Division {
   public static void main(String[] args) {
        int a, b, result;
          Scanner input = new Scanner(System.in);
        System.out.println("Input two integers");
          a = input.nextInt();
        b = input.nextInt();

          // try block
          try { result = a / b;
            System.out.println("Result = " + result);
        }

        // catch block
        catch (ArithmeticException e) {
            System.out.println("Exception caught: Division by zero.");
        }
   }
}
```

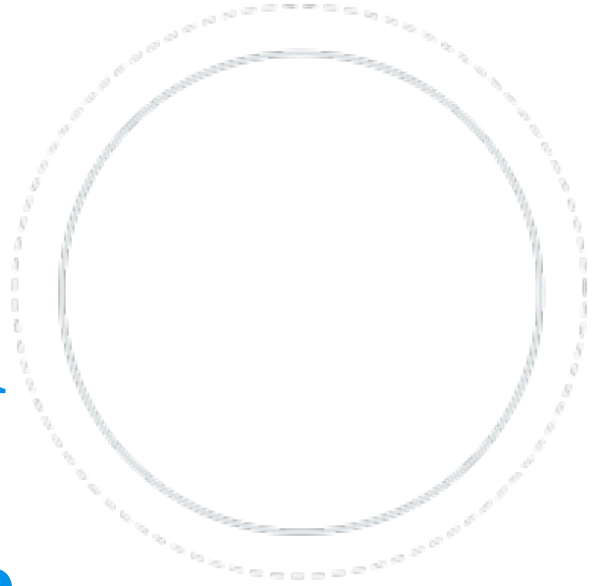
Pros & Cons, suggestions

- Don't mix error code and exception within single “module”
- Exception shouldn't leave “module”
- Log all information about error
- Don’t use top level exception handler for all unhandled exceptions
- Use exception, if additional hidden management is not an issue
- Develop hierarchy of exceptions during design time
- Define clear rules how to transform error codes into exceptions and vise versa



Compilers, Interpreters & Meta-programming

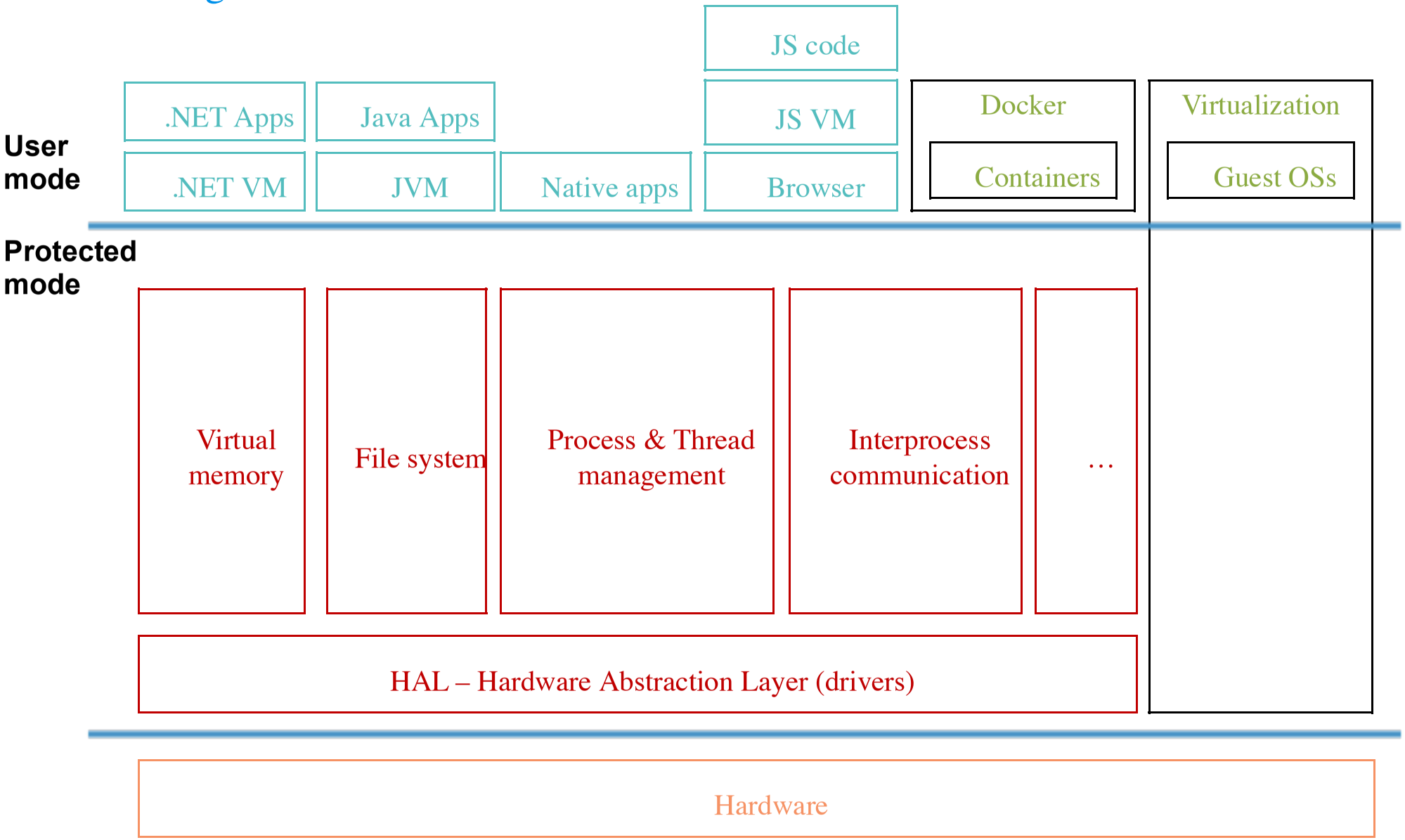
Multi threading, Client-Server, Peer2Peer



Virtualization

...

OS High-level structure



*Quotations are commonly printed as a **means of inspiration** and to invoke philosophical thoughts from the reader.*

This is a slide title

- Here you have a list of items
- And some text
- But remember not to overload your slides with content

You audience will listen to you or read the content, but won’t do both.

Big concept

Bring the attention of your audience over a key concept using icons or illustrations

You can also split your content

White
Is the color of milk and fresh snow, the color produced by the combination of all the colors of the visible spectrum.

Black
Is the color of coal, ebony, and of outer space. It is the darkest color, the result of the absence of or complete absorption of light.

In two or three columns

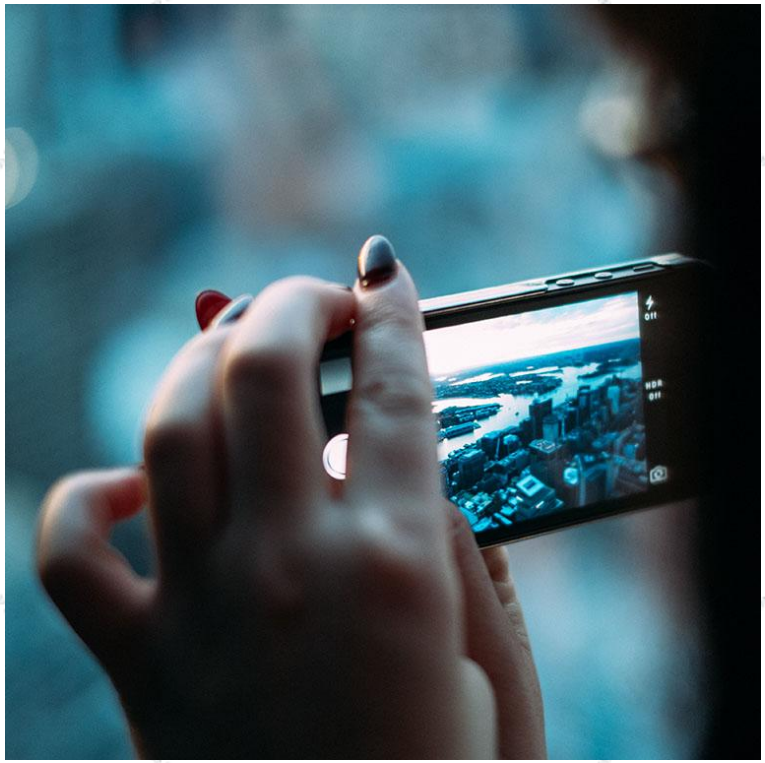
Yellow
Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.

Blue
Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.

Red
Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.

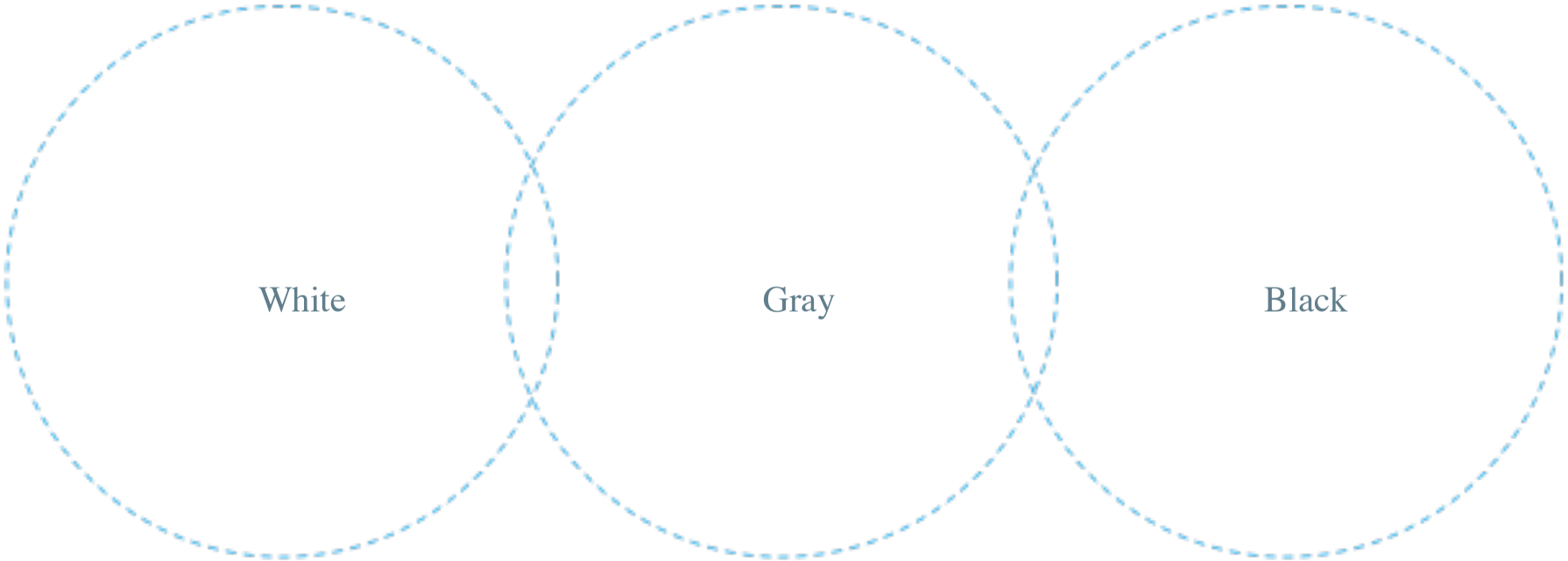
A picture is worth a thousand words

A complex idea can be conveyed with just a single still image, namely making it possible to absorb large amounts of data quickly.



Want big impact?

Use charts to explain your ideas



Or diagrams to explain complex ideas

Example text.

Go ahead and replace it with your own text. Go ahead and replace it with your own text. This is an example text. Go ahead and replace it with your own text. Go ahead and replace it with your own text. Go ahead and replace it with your own text.

Example text.

Go ahead and replace it with your own text. Go ahead and replace it with your own text. This is an example text. Go ahead and replace it with your own text. Go ahead and replace it with your own text. Go ahead and replace it with your own text.

Diagram featured by poweredtemplate.com

And tables to compare data

	A	B	C
Yellow	**10**	**20**	**7**
Blue	**30**	**15**	**10**
Orange	**5**	**24**	**16**

Maps

89,526,124

Whoa! That’s a big number, aren’t you proud?

89,526,124$

That’s a lot of money

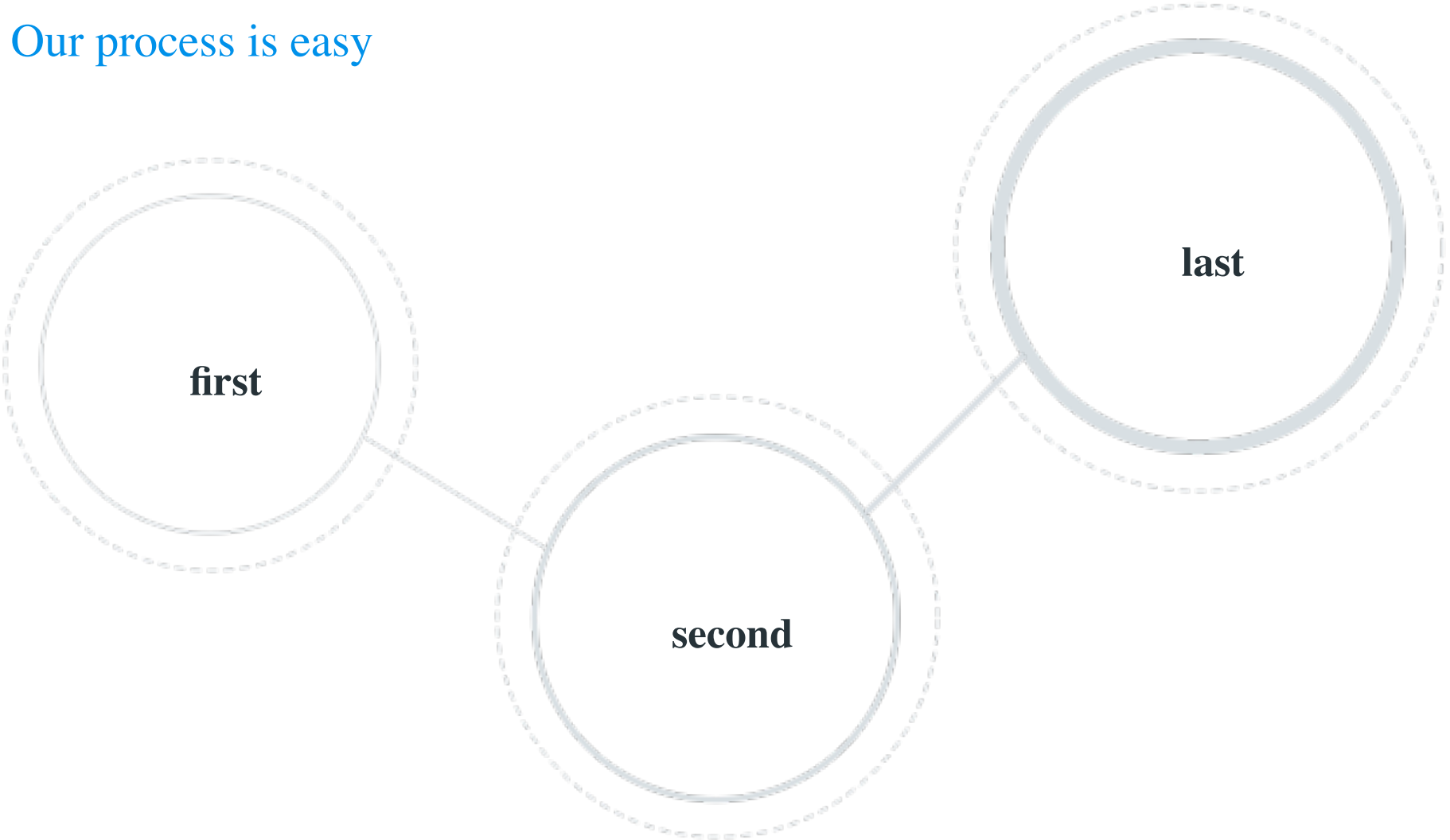
185,244 users

And a lot of users

100%

Total success!

Our process is easy



Let’s review some concepts

Yellow

Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.

Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.

Red

Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.

Yellow

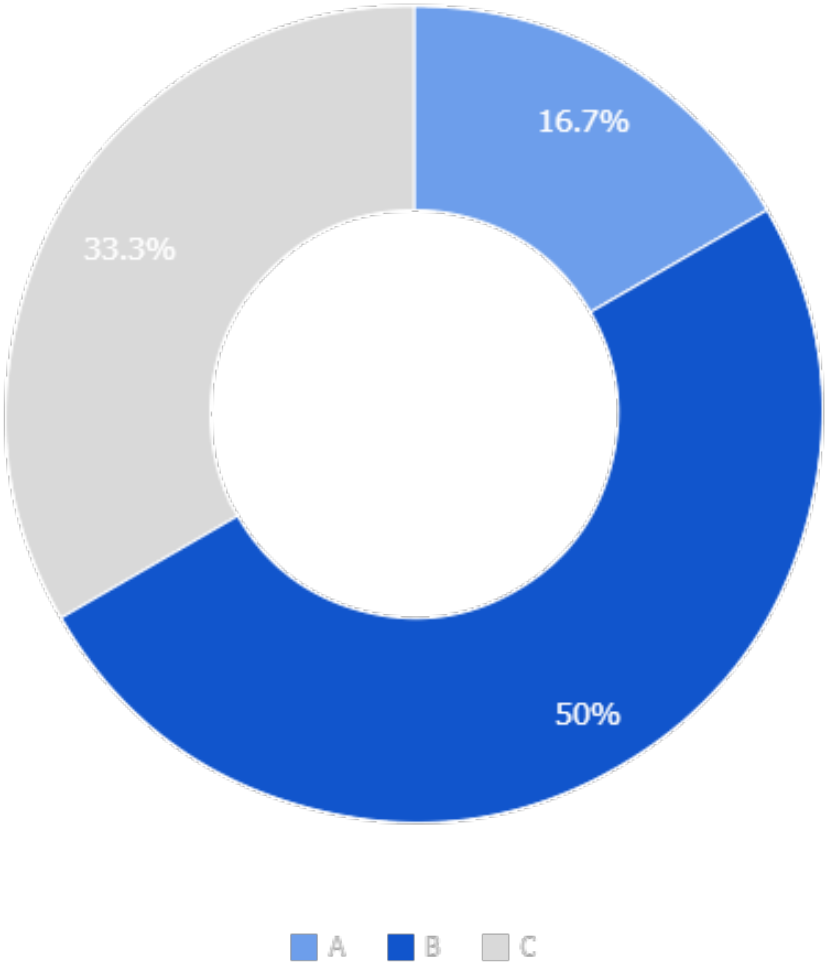
Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.

Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.

Red

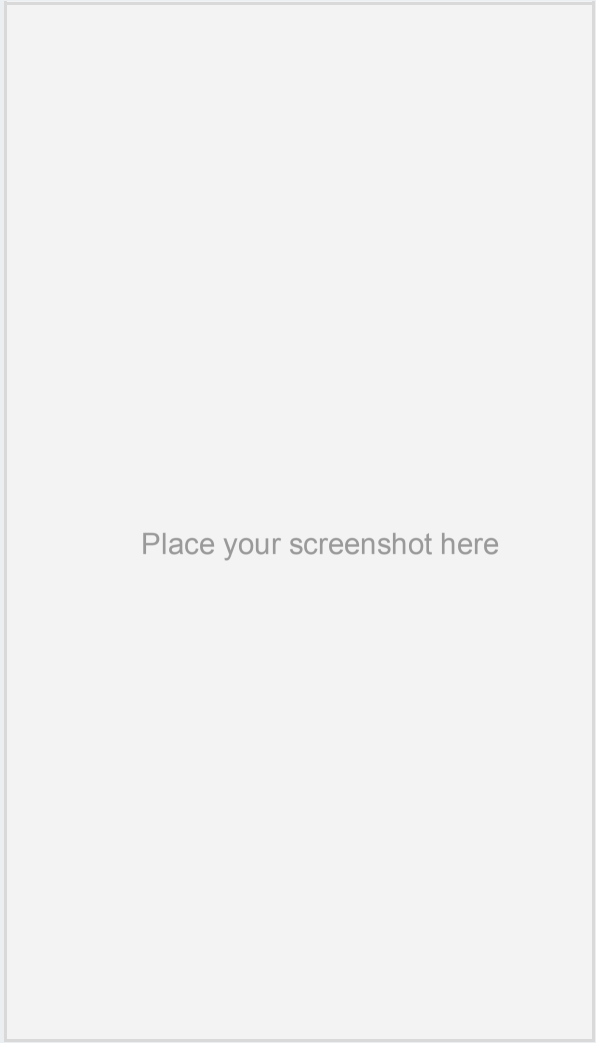
Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.



You can copy&paste graphs from Google Sheets

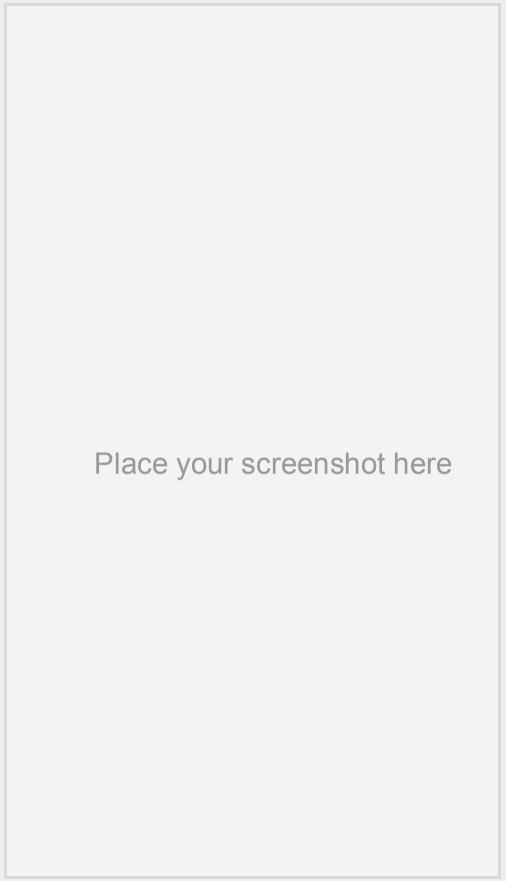
Android project

Show and explain your web, app or software projects using these gadget templates.



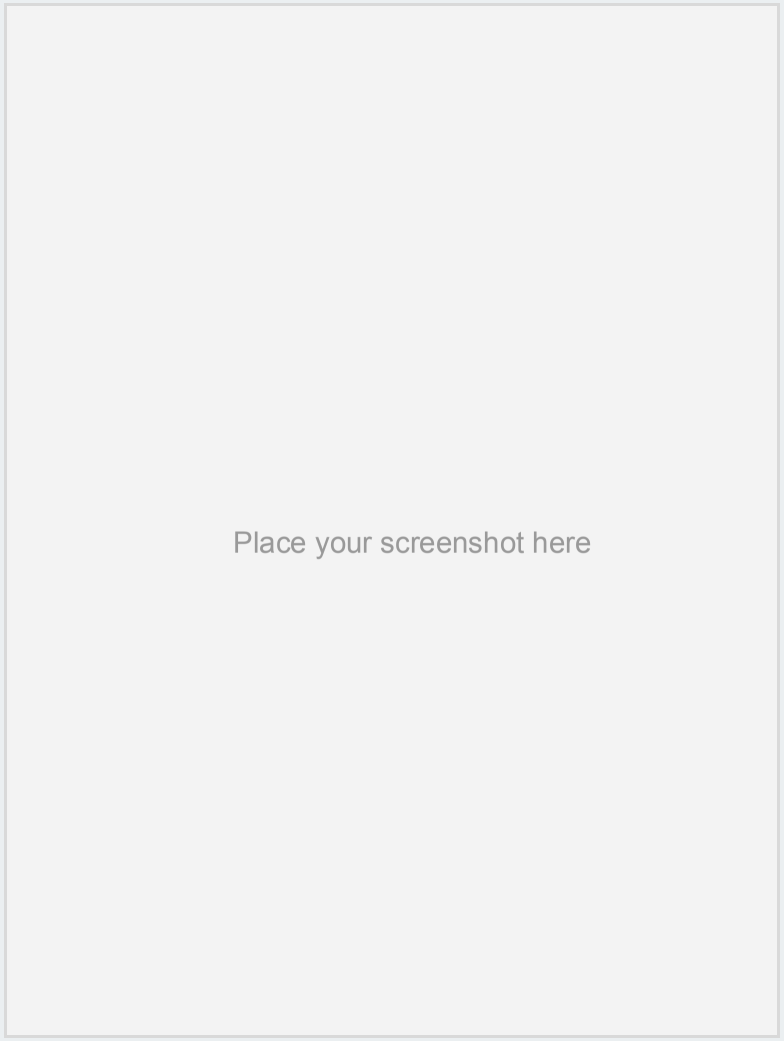
iPhone project

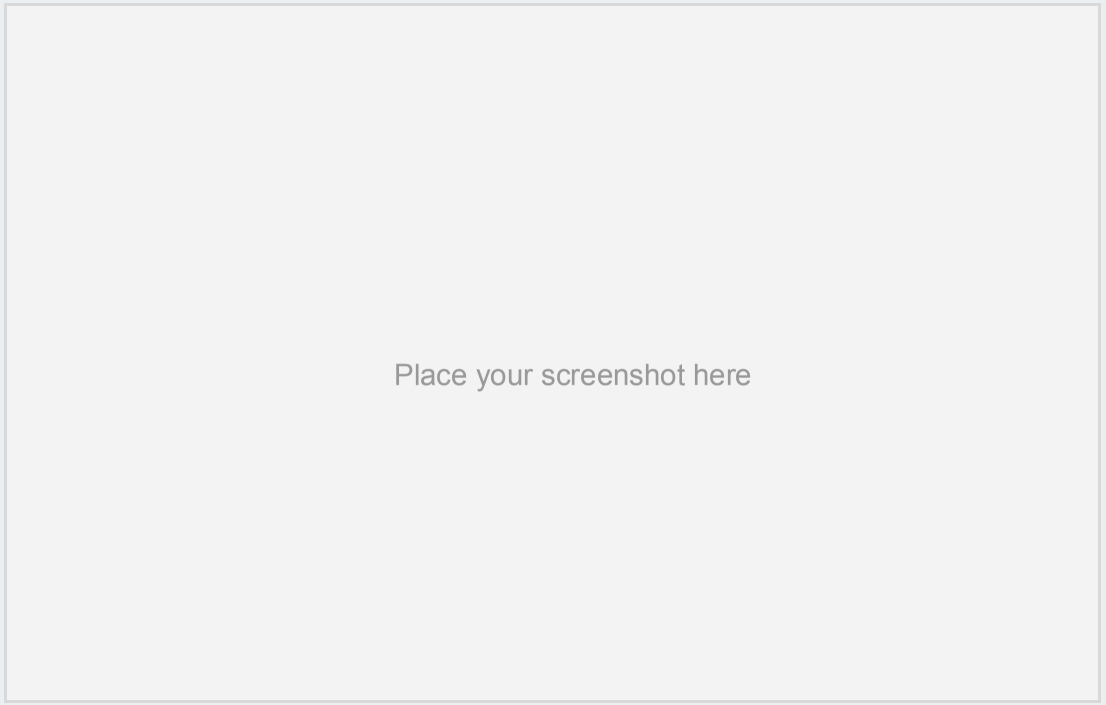
Show and explain your web, app or software projects using these gadget templates.



Tablet project

Show and explain your web, app or software projects using these gadget templates.





Desktop project

Show and explain your web, app or software projects using these gadget templates.

Thanks!

Any questions?

You can find me at:
@username
user@mail.me

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash & Death to the Stock Photo (license)

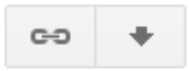
Presentation design

This presentations uses the following typographies and colors:

- Titles: **Roboto Slab**
- Body copy: **Source Sans Pro**

You can download the fonts on this page:
https://www.google.com/fonts#UsePlace:use/Collection:Source+Sans+Pro:400,700,400italic,700italic|Roboto+Slab:400,700

Click on the “arrow button” that appears on the top right



- Blue **#0091ea**
- Dark gray **#263238**
- Medium gray **#607d8b**
- Light gray **#cfd8dc**

You don’t need to keep this slide in your presentation. It’s only here to serve you as a design guide if you need to create new slides or download the fonts to edit the presentation in PowerPoint®

SlidesCarnival icons are editable shapes.

This means that you can:

- Resize them without losing quality.
- Change line color, width and style.

Isn’t that nice? :)

Examples:

Now you can use any emoji as an icon!
And of course it resizes without losing quality and you can change the color.

How? Follow Google instructions
https://twitter.com/googledocs/status/730087240156643328