ORDISOFTWARE™ ENGINEERING

AGILE CREATION OF OBJECT-ORIENTED APPLICATIONS

MANUFACTURING SOFTWARE GUIDELINES

METHODOLOGY & PROGRAMMING

VERSION 0.2

OLIVIER ROGIER

WWW.ORDISOFTWARE.COM

GITHUB.COM/ORDISOFTWARE/GUIDELINES

VERSION HISTORY

VERSION 0.1

- Created on October 2016.
- Released on 5 April 2018.

VERSION 0.2

- Upgraded on 5 April 2018.
- Since in French « développement » and « programmation » are synonyms but in English « development » means « engineering », documents were misspelled « development » and « methodology » and they are merged into « programming and methodology ».
- The first word of « manufacturing », « building » and « crafting » was chosen for the title because the second usually means « generating the machine code » during the production process and the latter is only recently used in our business.
- Released on 8 April.

TABLE OF CONTENT

LICENSE

FOREWORD

This document presents some development guidelines to produce libre, personal, private, commercial and military software.

They are a description of how the author tries to work currently. They are considerations coming from the practice of manufacturing own and business applications. They are generally basic and obvious. They are not absolutes and not something imposed as are arithmetic and geometry. They are malleable and improvable like lots of things in this wise human world.

Each computer practitioner as everyone has its own rules forged teacher after teacher, talk after talk, book after book, line after line, launch after launch, pixel after pixel, click after click, error after error, reboot after reboot and update after update.

Everyone mostly thinks having the best system, since it comes from learnings those work. Everyone think to have the best for self and for doing some things, each time this fact is thanked. Everyone sometimes just wants most of the time do tomorrow a better work than yesterday.

A programming system does not escape to the difficulty to work with others that have different means to do some things while improving each without making war to impose one while saying everyone is free to justify the denial of the existence of numbers and letters that are the sole cause of the reality created by the chromosomic intelligence of this area whose the first rule of any legal activity is democratically applicable for each to not willingly harm anybody.

DISCLAIMER

The author is not very advanced in the way of writing in English.

He was not able to learn to speak and write English properly, and not so much better French. But he knows well things like `start-stop`, `if-then-else` and `call-return`.

He uses a lot Google's online search engine and translator with English⇔French articles of Wikipedia and Wiktionary, as well as MS Word's linguistic tools.

He hopes that the reader will not hold against him for his way to express, for the tone he uses and for his mistakes.

WHO THIS DOCUMENT IS FOR

This document is for anyone who wants to know how the author uses computing technologies and tools to fabricate computer programs.

It mainly refers to Agile thoughts and C#.NET but it can be used with most of systems. It covers mechanisms related to structuring items of a project and elements of an application.

It may be enhanced as it allows well creating legal and allowed software that works properly.

HOW THIS DOCUMENT IS ORGANIZED

This document is divided in twelve parts:

- « *License* » specifies the terms of use for this document.
- « *Foreword* » presents this document and the author.
- « *Project tools* » indicates the means used by the author.
- « *Project documents* » indicates the types of notes produced for a project.
- « *Folder structure* » indicates how are organized the file elements of a project.
- « *Naming convention* » specifies the standards used to write the source code.
- « *Code formatting* » specifies the standards used to render the source code.
- « *Comments usage* » specifies the standards used to describe the source code.
- « *UI design* » indicates some user interface practices currently used by the author.
- « *Using Git* » indicates some rules currently used by the author.
- « *Using GitHub* » indicates some parameters currently used by the author.
- « *Bibliography* » indicates some books related to computers in the most recent version.

CONVENTIONS USED IN THIS DOCUMENT

Phrases use mainly the « French double angle quotes ».

The "Typewriter identical double quotes" is used to distinguish a technical thing.

A section that is intended to be written in a future release indicates:

This section is undescribed yet.

The mention of a computing artifact looks like:

`Menu / Submenu / Action`

`Filename.ext`

`www.domain.tld`

RGB colors are noted as: #000000

```
class Sample
{
}
```

- `Value 1`
- `Value 2`
- `Value n`

ABOUT LIBRE SOFTWARE

There are two categories of software: those which are proprietary and we must usually pay for their use, and those which are libre and we have no obligation to purchase.

Commercial and libre software are not necessarily opposable, and sometimes the objectives have no relationship with their differences that can be mixed according to the domain, the need, the type and the scope of a project.

In both cases, developers supply an immaterial work through a physical medium, for which they are intellectually the authors, and which takes time and investment. The purpose of a work being to live and survive, for oneself and for others, free software is not thus synonymous of gratuitous, unless it is a public service funded by taxes.

Donations are a source of income for free software designers. Just like we are free to use these programs, we are also free to define what we give according to our means. Sometimes the authors don't ask for money for various reasons.

But the shareware donation system, outside the case of amateurism regarding few currency units, except for humanitarian work not controlled by the State, is a false and problematical litigious solution for a false cash flow problem, and from a fiscal point of view it would be more accurate to consider the libre purchasing as commercial sales when the product is an intangible deliverable, which requires to not flat rate taxing the existence of an entity and only the generated flow of money.

Developing free or open source software is a vision about source code, sharing of knowledge and evolution of computing. You can read different points of view from:

- Free Software Foundation: `www.fsf.org/licensing/essays/free-sw.html`
- Open Source Initiative: `www.opensource.org/docs/osd`
- Creative Commons: `creativecommons.org/about/licenses`.

ABOUT THE AUTHOR

Olivier Rogier is a software craftsman mainly skilled in C#.NET and Delphi.

Such was the destiny of his abilities, of his will, of lived experience and of opportunities.

Despite constant unjustified and illegitimate oppressions and aggressions, he worked and works day and night every day when that is possible since his childhood for becoming and being a computer programmer, regardless of his results that were sometimes good and sometimes bad.

He was brought up with Basic, Assembler, C and C++ languages. His main aptitude is the object code, and the conceptualization of the data and its treatment.

When he was ten, the school has put a computer in his hands and one made him write a program on this machine equipped with a keyboard and a screen. One hour later, he said to himself that when he grew up, he would be a programmer.

As some of the first generation, he read some books and magazines. He read and reread them to know by heart the keywords of the language and to know how to control the elements of the machine. He entered by hand codes of little games and system hacks. Then he started writing his own programs. At first he bought a few games, then people from schools showed him how to copy the tapes to exchange them, and then they began to give themselves lots of software copies on floppy disks he accepted without knowing the value of the work.

The low secondary school guidance counselor told him that the best for him was to make an "IUT Informatique", and next an engineer school according to his results. He was entirely agreed even if he knew nothing about many things. But it did not go very well as planned and he did not follow the three quarters of courses. However, he had a very good teacher of analysis and design of information systems. Then his first project leader taught him everything there was to know in outline on his business and he has worked for major companies and big medical and financial organizations.

He now considers the right and need of the source code of all software sold or distributed free of charge, and therefore not falls within the internal and legal activity of a group nor the national security, to be as free and monetizable by its producer as the text of a book because of the immutable principle that a code hidden to the public is like a book hidden to the public.

To learn more about him:

- Twitter: `twitter.com/ordisoftware`
- Facebook: `www.facebook.com/ordisoftware`
- LinkedIn: `www.linkedin.com/in/ordisoftware`
- Contact: `www.ordisoftware.com/contact`
- Profile: `www.ordisoftware.com/about/author`
- Projects: `www.ordisoftware.com/projects`
- Blog: `www.ordisoftware.com/blog`
- Skills: `www.ordisoftware.com/business/skills`
- Achievements: `www.ordisoftware.com/business/history`
- Bibliography: `www.ordisoftware.com/business/bibliography`
- Service offer: `www.ordisoftware.com/services`

METHODOLOGY

AGILITY

OVERVIEW

Agile methods are the result of the practice and the afterthought from the use of methods called « traditional » that they incorporate and expand based on the following notions:

- *Iterative method*: the project is realized by compartments or portions, through the concepts of objects, components and packages.
- *Incremental method*: the project is realized by progression or refining, through the implementation of abstraction, polymorphism and genericity.

And:

- *Scenario*: these methods of production are based on unitary specifications of the functionalities that are derided into tasks or steps.
- *Deliverable*: a functional application is frequently and regularly builds to lead the advance of these methods, from the initial model until the last prototype that became the final software.

VALUES

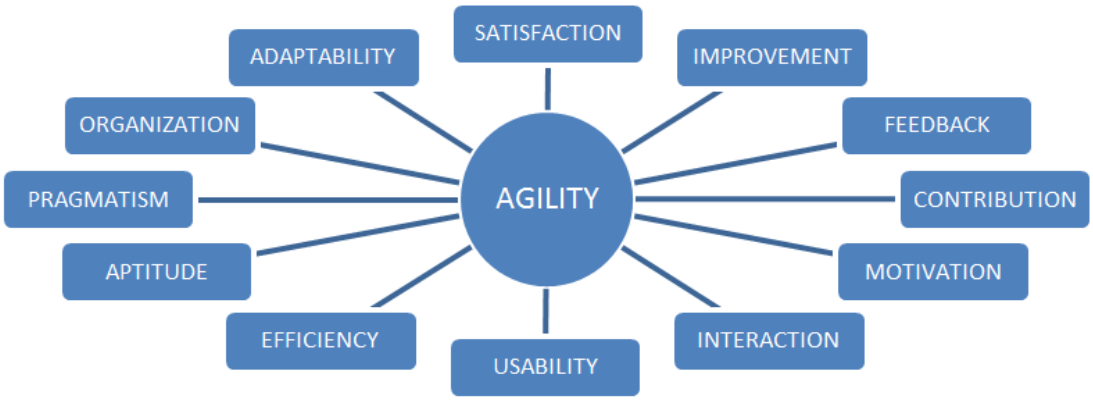
Agile methods rely on four basic values in order to master architectures:

- *Interaction*: communication has priority over methods and tools.
- *Result*: a program that works has priority over documentation.
- *Adaptability*: regular participation has priority over negotiations.
- *Improvement*: changing has priority over planning.

PRINCIPLES

These values are detailed in twelve principles:

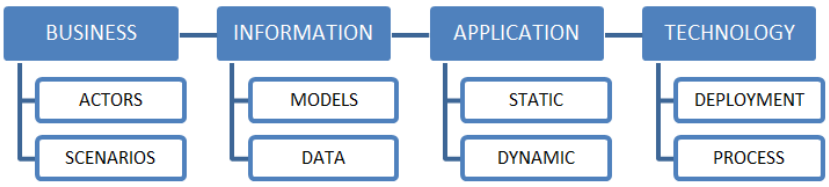
- *Satisfaction of the result:* priority is to deliver useful software to the user.
- *Improvement*: ability to change is a competitive advantage.
- *Feedback*: action is based on the regular delivery and the user response.
- *Contribution*: the different specialized contributors are regularly solicited.
- *Motivation*: environment and support are essential to success.
- *Interaction*: communication is the way of transmitting information.
- *Usability*: software that works is the indicator of progress.
- *Efficiency*: adopting a comfortable rhythm is the way to get the result.
- *Aptitude*: expertise and quality are continuously evaluated.
- *Pragmatism*: simplicity is even more essential that the project is complex.
- *Organization*: sharing of activities provides the best software.
- *Adaptability*: mutual and regularly introspection about the effectiveness adjusts the behavior of the team.



VIEWPOINTS

The project is usually approached from four considerations and five viewpoints:

- *Business* & *Use cases*: actors and scenarios.
- *Information* & *Design*: models and databases.
- *Application* & *Implementation*: static and dynamic aspects.
- *Technology* & *Deployment and process*: infrastructure and components.



DICHOTOMY

According to the Unified Process

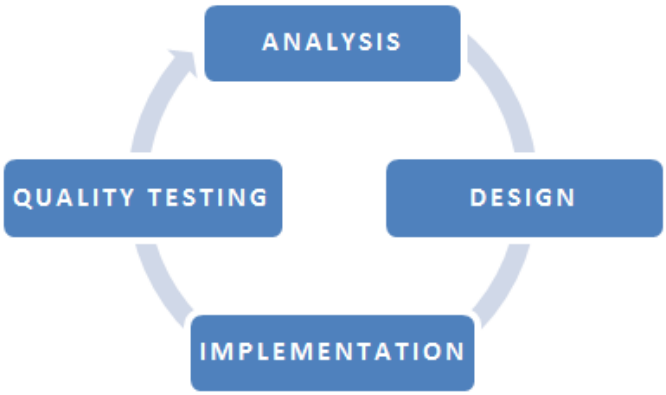
- *Inception* for *Initialization of the project*.
- *Elaboration* for *Analysis and high-level design*.
- *Construction* for *Low-level design and implementation*.
- *Transition* for *Quality testing and releasing*.

It is necessary to consider also a phase in its own right this stage:

- *Learning for issues related to Activities for training and technology intelligence.*

These phases of data-processing projects take place in four successive and retroactive steps that are imbricated one inside the other:

- The *analysis* defines « what to make » by specifying the technical elements to conceive to manage the entities and the data of your domain.
- The *design* defines « how to make » by identifying the items which make the program essential, as well as the choices to achieve them. Sketching the database and the user interface provides an indication of the tasks of development and the relevance of the selected solutions.
- The *implementation* is the writing of individual software components previously defined in an object-oriented language that provides quality, reliability, robustness, modularity, scalability and safety of the processings.
- The *quality testing* is an audit of the code, the checking the data coherence and the validation of the use of the program.



When there is no longer any step to perform, no action to be taken, that everything works smoothly and that the desired result is reached, the software is considered as finished and it is ready for deployment and allocation to the maintenance cycle.

DOCUMENTATION

GUIDELINES

Software guidelines are the rules that define how to create applications.

« Programming Guidelines » indicates technical and structural means used for the production.

« Methodology Guidelines » indicates executive and functional processes.

Some others can be made like for user interfaces that is included this guide, for user experience or for robots specificities.

They both take part in the « Manufacturing Software Guidelines » package.

They are used to establish documents related to a specific project.

The development guidelines should be used with consistency within an organization.

The methodology guidelines may vary depending on the needs.

The nomenclature set forth below is currently used by the author.

GLOBAL SPECIFICATION

This is the document for the project goals with legal-contract and links:

- The « Project Charter » describes for who, why and how exists the product.
  - Who are the project owners and users?
  - What is a simple and sketchy description of domains, problems and goals?
  - What is the overall direction of the project?
  - What are the papers to produce?
  - What is the first estimation of means and timings?

OVERALL REALIZATION

This is the documents set for the project implementation:

- The « Application Reference » describes how is constructed and deployed the program to achieve the « Project charter ».
- The « Designer Diagram Reference » describes the organization of models and packages.
- The « Developer Data Processing Reference » describes physical schemas of classes with development help files and database tables with generation scripts as well as code algorithms and procedures-triggers.
- The « UI Reference » describes how are managed the interactions between users and computers by using keyboard, mouse, phone, etc. and screens, windows, controls, etc.

HIGH-LEVEL DESIGN FOR FUNCTIONS

This is the documents set for the project elaboration with analysis and conceptual modeling:

- The « Use cases Reference » describes stories and diagrams that describe actors and scenarios acting on activities of the domain.
- The « Communication Reference » describes how actors exist as scenarios.
- The « Activity Reference » describes the dynamic view of use cases.
- The « Sequence Reference » describes how activities exist as scenarios.

LOW-LEVEL DESIGN FOR STRUCTURES

This is the documents set for the project construction with technical and physical modeling:

- The « Deployment Reference » describes how to install the product.
- The « Component Reference » describes the combination of components.
- The « Class Reference » describes abstraction of things from the domain.
- The « Object Reference » describes how class instances exist as living entities.
- The « Collaboration Reference » in describes how objects interact.
- The « State Reference » describes the comportment of objects according to scenario.
- The « DB Reference » describes tables and schemas if necessary.

USER DOCUMENTATION

This is the documents set for the users:

- The « User Manual » is the traditional installation and usage guide.
- The « Quick Start Guide » is the conventional summary of the user manual.
- The « Troubleshooting Reference » indicates what to do if the program does not do what the user want. It includes correcting the flow of operations in case of mistakes and actions to take in case of error message or even system crash.

TIME TRACKING STAGES

Any methodology acts on height main scopes over any dichotomy and nomenclature:

- The « Management » is the time to supervise the project.
- The « Training » is the time to learn things like skills and domains.
- The « Data » is the time to study and defining things like with a database.
- The « Processing » is the time to handle things like those in a database and UI.
- The « Manual » is the time spent to give instructions to users like with a guide.
- The « Setup » is the time to deliver the application to users like with an executable.
- The « Publicity » is the time to advertise potential users like with a public message.
- The « Support » is the time to help users in difficulty like with assistance or recycling.

MODELING

Whether thanked or represented, models precede, underlie, document and validate the production of software of quality.

The use of relational and object-oriented modeling brings simplicity, clarity and modularity in the conceptual representation of real things.

Ordisoftware™ usually uses the Entity–relationship model and the UML standard to specify and visualize structures, functions and interactions of systems.

To obtain the desired result, the modeling and the implementation are continuously set in correspondence.

The constant review of models and code, associated with the refactoring, are essential methods to success.

ECOSYSTEM

As the mutation of the classical programming to object-oriented programming has taken time to mature, since the creation of punch cards, many Agile methods are developed based on the sensitivity of their creators and depending on industrial requirements.

- *Rapid Application Development* (1991)*: Based on an analysis-design-construction iterative cycle to build a deliverable every 3-4 months by independent teams.
- *Unified Process* (1996)*: Based on use cases, focused on UML architectural views, driven by iterative and incremental methods.
- *Extreme Programming* (1999)*: Based on the construction of the application, with very short delivery cycles, the privileged integration of the customer into the team, and the use of specific coding techniques (simplicity, refactoring, conventions, common vocabulary, unit testing, pair programming, shared code, continuous integration, respect of reality and constraints).
- *Object-oriented applications analysis and design method* (2003)*: UP simplification associated with RAD focused on GUI and UML to define the structures and functions of the system and to achieve an incremental application prototyping.
- *Kanban* (2010)*: Inspired by Lean for the process management, focused on the organization, the communication and the knowledge.
- *Dynamic Systems Development Method* (1995): Structured development cycle by extension of RAD, with a higher frequency of delivery controlled by tests.
- *Feature Driven Development* (1999): Similar to RAD, with priority to features that deliver value, and the use of five activities (develop overall model, build feature list, plan by feature, design by feature, build by feature).

- *Scrum* (2001): Based on the goal and the complexity of the project goal according to the philosophy of rugby.
- *Lean Software Development* (2003): Based on eliminating waste, learning, on quality, fast return of delivery, later decision making, power given to the team, and overall vision.
- *Crystal Clear* (2004): Based on communication and collaboration, for small projects.

** Methods used preferentially by the author*

PROGRAMMING

TOOLS

The author currently uses the following tools to work on an assembled midrange PC.

These was a selection of what he personally found actually the best for him.

OPERATING SYSTEM

Windows @ `www.microsoft.com/windows`

FILE MANAGER

Total Commander @ `www.ghisler.com`

BACKUP MANAGER

O&O DiskImage @ `www.oo-software.com`

Macrium Reflect @ `www.macrium.com`

FreeFileSync @ `www.freefilesync.org`

AutoVer @ `autover.codeplex.com`

SOURCE CONTROL

Git @ `git-scm.com`

GitHub @ `github.com`

TortoiseGit @ `tortoisegit.org`

Git Extensions @ `gitextensions.github.io`

TEXT EDITOR

Notepad2-mod @ `xhmikosr.io/notepad2-mod`

WORD PROCESSOR

Word @ `products.office.com/word`

PDF files are generated with the following options:

- *ISO 19005-1 compliant (PDF/A).*
- *Document structure tags for accessibility.*

SPREADSHEET

Excel @ `products.office.com/excel`

IMAGE PROCESSOR

XnView @ www.xnview.com

Axialis Icon Workshop @ www.axialis.com/iconworkshop

GIMP @ www.gimp.org

DIAGRAM DESIGNER

Software Ideas Modeler @ www.softwareideas.net

AGILE STORYBOARD

ZenHub @ www.zenhub.com

TIME TRACKING

AllNetic Working Time Tracker @ www.allnetic.com

INTEGRATED DEVELOPMENT ENVIRONMENT

Visual Studio @ www.visualstudio.com

NuSphere PhpED @ www.nusphere.com/products/phped.htm

Visual Studio Extensions

GitHub Extension for Visual Studio @ visualstudio.github.com

TGit @ github.com/sboulema/TGIT

GitExtensions VSIX @ gitextensions.github.io

Codinion @ www.codinion.com

CodeMaid @ www.codemaid.net

Power Commands @ github.com/Microsoft/VS-PPT

Editor Guidelines @ github.com/pharring/EditorGuidelines

Solution Error Filter @ github.com/Microsoft/VS-PPT

File Icons @ github.com/madskristensen/FileIcons

Markdown Editor @ github.com/madskristensen/MarkdownEditor

Disable Solution Dynamic Nodes @ github.com/madskristensen/ToggleFeatures

Editor ToolTips @ github.com/Oceanware/TameVisualStudioEditorToolTips

Hide Suggestion @ marketplace.visualstudio.com/items?itemName=...

DATABASE

SQL Server @ `www.microsoft.com/sql-server`

SQLite @ `www.sqlite.org`

SQLite.NET @ `system.data.sqlite.org`

SQLite Expert @ `www.sqliteexpert.com`

DbSchema @ `www.dbschema.com`

COMMENTS GENERATOR

Atomineer Pro Documentation @ `www.atomineerutils.com`

DOCUMENTATION GENERATOR

Sandcastle Help File Builder @ `github.com/EWSoftware/SHFB`

SETUP PACKAGER

Inno Setup Installer @ `www.jrsoftware.org/isinfo.php`

PATHS

This section is undescribed yet.

NAMING

FILES

This section is undescribed yet.

NAMESPACES

This section is undescribed yet.

TYPES

This section is undescribed yet.

ENUM

This section is undescribed yet.

CLASS

This section is undescribed yet.

INTERFACE

This section is undescribed yet.

VARIABLES

This section is undescribed yet.

INSTANCE

This section is undescribed yet.

LOCAL

This section is undescribed yet.

METHODS

This section is undescribed yet.

COMMENTS

FILES

This section is undescribed yet.

NAMESPACES

This section is undescribed yet.

TYPES

This section is undescribed yet.

VARIABLES

This section is undescribed yet.

INSTANCE

This section is undescribed yet.

METHOD

This section is undescribed yet.

ALGORITHMS

This section is undescribed yet.

FORMATTING

INDENTATIONS

This section is undescribed yet.

LINES

This section is undescribed yet.

BRACKETS

This section is undescribed yet.

DECLARATIONS

This section is undescribed yet.

SIGNATURES

This section is undescribed yet.

STATEMENTS

This section is undescribed yet.

ALLOCATIONS

This section is undescribed yet.

UI DESIGN

CONSOLE

This section is undescribed yet.

FORMS

This section is undescribed yet.

WEB

This section is undescribed yet.

MOBILE

This section is undescribed yet.

TV

This section is undescribed yet.

USING GIT AND GITHUB

NAMING ARTIFACTS

REPOSITORY

<project-name>

Examples: Core-Library

BRANCHS

Any combination like:

- <issue-group>/<issue-type>/<issue-item(/<summary>)(/<issue-id>)
- <issue-group-or-type>/<issue-item>(/<summary>)(/<issue-id>)
- <issue-group-or-type>/<issue-item-and-or-summary>(/<issue-id>)

Examples:

- design/method/text/markdown/#100
- bug/install/icons-desktop/#45
- test/ui-db-settings

TAGS

<version-or-stage>

Examples:

- v0.1
- v1.2.3
- v2.0.0-rc0

COMMITS

The seven rules from `chris.beams.io/posts/git-commit`:

- *Separate subject from body with a blank line.*
- *Limit the subject line to 50 characters.*
- *Capitalize the subject line.*
- *Do not end the subject line with a period.*
- *Use the imperative mood in the subject line.*
- *Wrap the body at 72 characters.*
- *Use the body to explain what and why vs how.*

Common commits actions are:

- Add, Rename, Remove, Delete.
- Set, Update, Change, Improve, Fix, Move.
- Generate, Clean, Refactor, Rework.
- Initial commit, Merge, Release.

A domain can be specified by using an issue-item token:

```
ui: Fix the main form size
db: Add a script to create a table
manual: Update thefile.html
```

MILESTONES

Milestones allow identifying project big steps as agility and UP process : Inception, Elaboration, Construction and Transition.

For simple or non-software projects such as this guide milestones can be:

- *Version 1*
- *Version 2*

ISSUE LABELS

EPIC

ZenHub allows using special stories called Epic to gather other stories.

Color is Dark Blue #3E4B9E.

GROUP

Group defines the area concerned by the issue.

Color is Teal #006B75.

```
group: project (management)
group: training (learning)
group: analysis (requirements gathering)
group: design (modeling)
group: code (implementation)
group: manual (documentation and guide)
group: deploy (setup and migration)
group: user (assistance and communication)
```

TYPE

Type defines the gender of the issue.

Color is Green #0E8A16.

```
type: legal (license)
type: layout (organization and planning)
type: method (guideline)
type: admin (supervision)
type: feature (functionality)
type: improve (extend feature)
type: check (test, revision and validation)
type: bug (error)
type: feedback (reaction)
```

ITEM

Item defines the thing affected by the issue.

Color is Blue #1D76DB.

```
item: app (product and executable)
item: diagram (representation)
item: data (information)
item: source (code file)
item: install (packager)
item: text (writing)
item: tool (third party software)
item: ui (user interface)
item: ux (user experience)
item: other
```

PRIORITY

There is no medium priority since it is a loss of time to set and read it.

Thus it is easy to see the cards with low or high priority and others are ordinary.

```
prio: critical [Dark Red #900000]
prio: high [Red #CA2525]
prio: low [Dark Cyan #BFDADC]
```

IN PROGRESS

In progress defines an issue being solved and it is used in conjunction with some State label.

Color is Yellow #FFD700.

STATE

State indicates the progress of the work not towards the time but the remaining tasks.

Six points of a Gaussian curve are used to estimate the `In progress` pipeline.

This percentage is not about time because the tens first and last parts are generally longer while at the middle the things can be very fast:

- *When the task starts there is no really competence and no good visibility.*
  It has not started because it is taking its place to run on the racetrack.
  Things often seem to be simple and easy even for big task.
  It is not uncommon to spend a quarter of the time on this inception phase.
- *When the task comes to its end, there is a need to begin checking that all is really fine.*
  It is not running anymore because and it is shutting down on the racetrack.
  Things are more complex and more interactive even they look effortless.
  This transition phase can sometimes be more half the time.

This percentage may be reevaluated according to addition or cancellation of the complexity.

Color is Yellow #FFD700.

```
state: todo (selected) [Pale Green #C2E0C6]
state: delayed (deferred) [Gray #CACACA]
state: cancelled (abandoned) [Light Gray #EAEAEA]
state: moved (to another project) [Light Gray #EAEAEA]
state: wontfix (failed) [Dark Gray #707070]
state: 10% (work started)
state: 25%
state: 50%
state: 75%
state: 90% (almost completed)
state: 100% (done) [Light Yellow #FFF3B5]
```

ZENHUB BOARDING

One GitHub project can be used as a storyboard for one or more use case diagrams.

While this not allows advanced management yet, the author uses ZenHub and Epics.

PIPELINES

ZenHub Pipelines allow setting the stage of issues like on a Kanban workflow board.

```
Incoming
Icebox
Backlog
Epics planned
Epics live
To do
In progress
Done
Ratifying
Closed
```

```
New issues
Delayed
To consider
Big goals intended
Big goals in progress
Sub or indivudial goals to do
In progress at (0)10-90%
Not in progress but at 100%
To validate
Closed
```

On small projects Incoming, Icebox, Backlog, Done and Ratifying can be omitted, and Epics planned and live pipelines can be one while the in progress label indicates the difference.

Epics allow distinguishing Featured User Stories from Action User Stories.

Visitors of the project's page that are not connected to an account extended with ZenHub can't see this layering yet and only `in progress` label and `closed` issues can be used to distinguish them from other, so manual labels must be used in addition to moving cards.

ISSUE AS USER STORY

An Issue is used as a user story by indicating its description containing tasks in checklist.

```
[Issue #1] Prepare the repository

           As a developer,
           I want to establish the repository,
           so I can construct the software.

           ☐ Create the repository
           ☐ Setup the repository
           ☐ Specify the license
```

ISSUE ESTIMATE

Estimate field is used to define the issue complexity from `1` to `5` or to `10` for example, by considering knowledge, competence, technicity and range required by the issue.

Epic Issue estimation is usually not done because it is finished when all linked issues are finished and this value can be viewed on Issue details in the panel added by ZenHub and the Epic points divided by the issues count rounded to the upper is thus used as an estimate.

Estimate time is out of scope of this document and falls under any appropriate methodology, but timings can be defined and adjusted using the burndown chart as the project progress.

ISSUES HIERARCHY

Visitors of the project's page that are not connected to an account extended with ZenHub can't see this design yet without check-listing sub-issues in the description.

FEATURED USER STORY AS A HIGH-GOAL THAT ENCAPSULATES LOW-GOALS

An Issue is used as a complex story containing references to other issues by using ZenHub Epic label.

It should contain a checklist of all sub-issues as high-tasks.

```
[Epic Issue #1]  Prepare the repository

                 ☐ Create the repository #2
                 ☐ Setup the repository #3
                 ☐ Specify the license #4
```

ACTION USER STORY AS LOW-GOAL TO ACHIEVE HIGH-GOAL

An Issue is used as a simple story acting as a card of what a user want by attaching it to an Epic Issue.

```
[Issue #2] Create the repository

          ☐ Add a repository
          ☐ Create a first branch
```

```
 [Issue #3] Setup the repository

          ☐ Define Milestones
          ☐ Define Labels
```

```
[Issue #4] Specify the license

          ☐ Examine available licenses
          ☐ Choose a license
          ☐ Publish the LICENSE file
```

BIBLIOGRAPHY

THOMSON COMPUTERS

Manuel technique du TO7-70 (Cedic/Nathan 1984)
Initiation et Référence du Basic TO7-70 (Cedic/Nathan 1984)
Initiation et Référence du Logo TO7-70 (Cedic/Nathan 1984)
Manuel de l'Assembleur 6809 du TO7-70 (Cedic/Nathan 1984)
Le Basic Q-D.O.S. du TO7-70 (Cedic/Nathan 1985)
Guide du TO9 (Cedic/Nathan 1985)
Guide du Basic TO9 (Cedic/Nathan 1985)
Super jeux MO5 et TO7-70 (Jean-François Sehan - PSI 1985)
Pratique du TO7-70, programmation niveau 1 (Henri Lilen - Editions Radio 1984)
Pratique du TO7-70, programmation niveau 2 (Henri Lilen - Editions Radio 1984)
50 programmes assembleur T07-70 (B. Geoffrion, R. Weiss - Editions Radio 1985)

PC MICROPROCESSORS AND SYSTEMS

8088 Assembleur (Henri Lilen - Editions Radio 1986)
8088 et ses périphériques (Henri Lilen - Editions Radio 1986)
Cours pratique de logique pour microprocesseur (Henri Lilen - Editions Radio 1986)
Cours fondamental des microprocesseurs (Henri Lilen - Editions Radio 1987)
Microprocesseurs (Henri Lilen - Dunod 1995)
PC system programming for developers (Michaël Tischer, Data Becker 1989)
PC Interdit (Bertelsons, Rasch, Hoff - Micro Application 1995)
Mastering Turbo Assembler (Tom Swan - Sams 1995)
8086 to 80486 Instruction Set Reference Guide
80386 Programmer's Reference Manual (Intel 1986)
x86 Developer’s Manual Volume 1: Basic Architecture (Intel 1997)
x86 Developer’s Manual Volume 2: Instruction Set Reference (Intel 1997)
x86 Developer’s Manual Volume 3: System Programming Guide (Intel 1997)
Protected mode software architecture (Tom Shanley - Addison Wesley 1996)
Operating System concepts (Silberschatz Galvin - Addison Wesley 1998)
Programmation des API Win32 (Simon, Gouker, Barnes - S&SM 1998)
VBScript Programmer's Reference (A. & K. Kingsley-Hughes, D. Read - Wrox 2007)

BORLAND IDES

Turbo Basic (Borland 1988)
Turbo Pascal (Borland 1988)
Turbo C++ (Borland 1989)
Turbo Prolog (Borland 1989)
Delphi 5 Objet Pascal Language Guide (Borland 1999)
Delphi 5 Developer's Guide (Borland 1999)
Mastering Delphi 2 (Marco Cantu - Sybex 1996)

C AND C++ LANGUAGES

Guide SOS du Turbo C (Jörg Schieb - Micro Application 1990)
Le langage C++ (Livret de cours IUT Informatique 1996-1997)
Visual C++ 5 (Christian Fleischhauer - Micro Application 1997)
C++ Primer Plus (The waite group's - Sams 1998)
Effective C++ (Scott Meyers - Addison Wesley 1996)
More effective C++ (Scott Meyers - Addison Wesley 1997)
The design and evolution of C++ (Bjarne Stroustrup - Addison Wesley 1994)

C# AND .NET FRAMEWORK

The C# language (Hejlsberg, Wiltamuth, Golde - Addison Wesley 2004)
Component-based development with Visual C# (Ted Faison - M&T Books 2002)
Professional .NET framework (Collective - Wrox 2001)
Beginning C# 2005 Databases (Karli Watson - Wrox 2006)
Beginning Visual C# 2008 (Collective - Wrox 2008)
Professional C# 2008 (Collective - Wrox 2008)
Professional .NET 2.0 Generics (Tod Golding - Wrox 2005)
Professional ADO.NET 3.5 with LINQ and Entity Framework (Roger Jennings - Wrox 2009)
Professional Visual Studio Extensibility (Keyvan Nayyeri - Wrox 2008)
Professional Refactoring in C# & ASP.NET (Danijel Arsenovski - Wrox 2009)
Professional Test Driven Development with C# (J. Bender, J. McWherter - Wrox 2011)
C# Design and Development Expert One on One (John Paul Mueller - Wrox 2009)
C# 3.0 Design Patterns (Judith Bishop - O'Reilley 2008)
C# 3.0 Cookbook (Jay Hilyard - O'Reilley 2008)
C# 4.0 How-To (Ben Watson - SAMS 2010)

JAVA AND WEBPAGES

Pure Java 2 (Kenneth Litwak - Sams 1999)
Using XHTML, XML & Java 2 (Eric Ladd, Jim O'Donnell - Que 1999)
Professional WordPress (Hal Stern, David Damstra, Brad Williams - Wrox 2010)
Professional WordPress Plugin (Brad Williams, Ozh Richard, Justin Tadlock - Wrox 2011)
Professional JavaScript for Web Developers (Nicholas C. Zakas, Wrox 2012)
Beginning HTML and CSS (Rob Larsen - Wrox 2013)

DATABASES AND SQL

Oracle et SQL (Livret de cours IUT Informatique 1996-1997)
Oracle 8 (Roger Chapuis - Dunes-Laser 1998)
PHP4 & MySql (G.A. Leierer, R. Stoll - Micro Application 2000)
The SQL Guide to SQLite (Rick F. van der Lans - Lulu 2009)
Beginning SQL Server 2008 Programming (Robert Vieira - Wrox 2006)
Beginning Database Design Solutions (Rod Stephens - Wrox 2008)
Beginning XML (Joe Fawcett, Liam R. E. Quin, Danny Ayers - Wrox 2012)
GitHub Essentials (Achilleas Pipinellis - Packt Publishing Limited 2015)
Professional Git (Brent Laster - Wrox 2016)
Excel 2007 (Collective - Eni Editions 2007)

ALGORITHMIC AND ARTIFICIAL INTELLIGENCE

Programmation structurée en BASIC (Francis Crochet - Editions Radio 1987)
Essential Computer Mathematics (Seymour Lipschutz - McGraw-Hill 1985)
Programmation récursive (Livret de cours IUT Informatique 1996-1997)
Programmation des listes chainées (Livret de cours IUT Informatique 1996-1997)
Programmation des graphes (Livret de cours IUT Informatique 1996-1997)
Procedural Elements of Computer Graphics (David Rogers - McGraw-Hill 1988)
Computer graphics (Foley, van Dam, Feiner, Hughes - Addison Wesley 1997)
Virtual reality excursions in C (Watkins, Marenka - AP Professionnal 1994)
Mathématiques pour l'informatique (Collective - Dunod 2008)
Logic Programming with Prolog (M.A. Bramer - Springer London Ltd 2005)
L'Intelligence Artificielle pour les développeurs C# (Virginie Mathivet - Eni Editions 2017)
Essential Algorithms (Rod Stephens - Wiley 2013)

SOFTWARE DEVELOPMENT

Le génie logiciel (Livret de cours IUT Informatique 1996-1997)
The UML User Guide (Booch, Rumbaugh, Jacobson - Addison-Wesley 2001)
Extreme Programming (Chromatic - O'Reilly 2005)
Practices of an Agile Developer (Subramaniam & Hunt - Pragmatic Bookshelf 2006)
Code Leader (Patrick Cauldwell - Wrox 2008)
Beginning Software Engineering (Rod Stephens - Wrox 2015)
Domain-Driven Design (Scott Millett, Nick Tune - Wrox 2015)
Méthode orientée-objet intégrale MACAO (Jean-Bernard Crampes - Ellipses 2003)
Agile principles, patterns and practices in C# (R. C. & M. Martin - Prentice Hall 2006)
Clean code (Robert C. Martin - Prentice Hall 2008)
Clean Architecture (Robert C. Martin - Prentice Hall 2017)
The clean coder (Robert C. Martin - Prentice Hall 2011)
The Software Craftsman (Sandro Mancuso - Prentice Hall 2014)
Code complete (Steve McConnell - Microsoft Press 2004)
Ergonomie des interfaces (Jean-François Nogier - Dunod 2011)
Producing Open Source Software (Karl Fogel - O'Reilly Media 2005)
Word 2010 (Collective - Eni Editions 2010)

OTHERS TOPICS

Tennis (Nathan 1986)
Eat to win (Robert Haas - Scribner 1985)
Blood and guts (Dorian Yates - 1993)
Relativity (Albert Einstein - Digireads 1990)
What is life (Erwin Schrödinger - Cambridge University Press 1992)
Light and Matter (Richard Feynman - Princeton University Press - Seuil 1985)
Histologie (Jacques Poirier, Jean-Louis Ribadeau Dumas - Masson Abrégé 1988)
Cours de biologie cellulaire (Pierre Cau, Raymond Seïte - Ellipses 2009)
Dictionnaire de la Bible Hébraïque (Marchand Ennery - Colbo 1996)
Les bases de l'harmonie (Philippe Ganter - Dareios IDMusic 2007)
Neuromarketing (Patrick Renvoisé, Christophe Morin - SalesBrain Publishing 2005)
The Mental Game of Poker (Jared Tendler - 2011)