Creating a software product

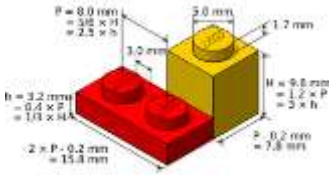
2014-01-22 – acaldmail@gmail.com

Creating software product

- Gathering requirements and writing a specification
  - Product description
  - Use-case
  - Deployment and maintenance scenarios
- Creating the project
- Setup of the design team based on
  - Competence
  - Expertise
  - Attitudes
- Resolving other organizational aspects
  - Logistics
  - Formal and legal issues

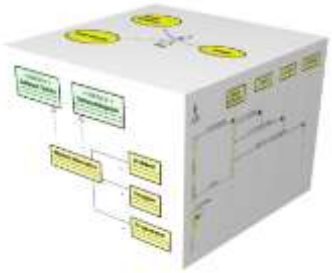
From requirements to design

- The requirements analysis produces
  - A proposal draft to discuss with the client, that produces, at the end of one or more iterations:
    - Estimation of resources to be allocated and the skills required
    - Estimation of effort and time
    - Acceptance criteria and test planning
    - Criteria for versioning and deployment
    - Criteria for managing the maintenance and support
    - User and project documentation



The project includes the design of ...

- Components and iterations with the target system (HW and SW)
- Type of UI and other user's iterations
- Monitoring / logging / debugging
- Security / Resilience
- Upgradability / Scalability
- Dependencies on third-party libraries (including the licensing)
- Delivery, versioning and software license



Tools

The tools needed are identified during the analysis phase include

- Hardware devices
- Software applications such as languages, libraries, development tools, simulators, etc.
- Backup and versioning software
- Communication systems (HTTP server, Email server, remote control, chatting, VoIP, etc.)

Creation of development team

Definition of roles and responsibilities in the team must be made on the basis of aptitude and skill.

- Motivation
  - The success of a project and its quality are the result of the joint effort and commitment of motivated people
- Rules and definition of WoW
  - They also serve to define choices and methods in order to make more uniform and simple exchange of information
- Communication and sharing of knowledge
  - The communication must be clear and based on the respect of the people even before the roles.
  - The sharing of knowledge is essential to have a vision of the goals to achieve

Before starting the project

The team and the project manager collaborate to

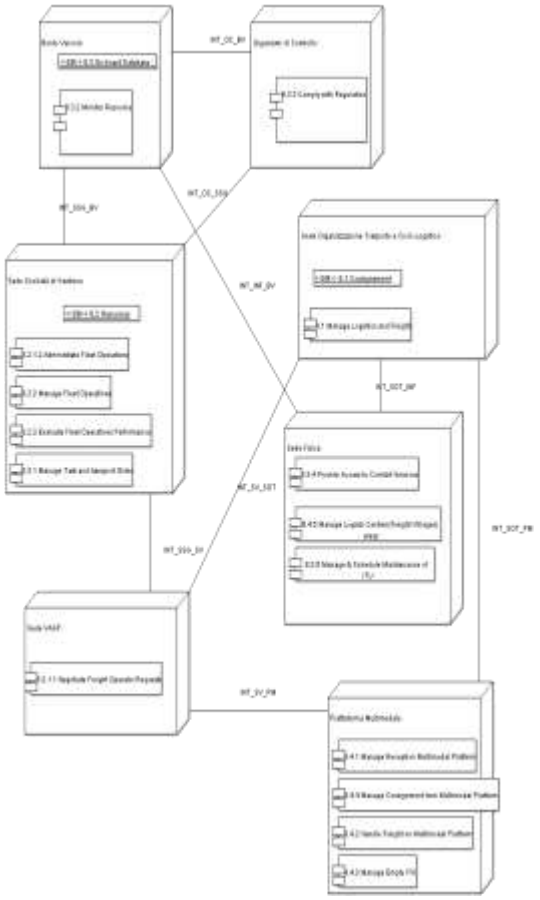
- Plan of further phases of analysis and design, including also
  - Prototyping
  - Definition of the user and project documentation
  - Acquisition of skills
- Schedule of activities
  - Plan details shared and viable
  - Schedule of testing and demo to the customer
  - Definition of acceptance criteria
  - HW and SW resources allocation
  - Resolution of the issues related to non-technical aspects

It begins with the design of macro blocks of the "software project"

The information necessary to identify the macro blocks of the product comes from the requirements.

The system can be decomposed in different views/layers, including:

- Drivers, Programs, Services
- IPC and networking services
- (G) UI
- Middleware, 3PP Libraries, modules and tools
- Iterations and usage scenarios
- The delivery and upgrade scenarios
- Debugging and monitoring methods



The choice of languages and development tools

- The choice of the software factory depends on many factors, among the most important we note:
  - HW / SW target (portability on different O/S)
  - Type of component to implement (user space application, services, drivers in kernel space, firmware, UI, GUI, etc ...)
  - Resiliency features and security requirements
  - Modularity, interoperability and scalability
  - Performance and realtimeness
  - Availability of libraries, tools, and third-party modules
  - Availability tools such as compilers, profilers, debuggers and IDEs
  - Expertise and experience of team members
  - HW tools availability
  - Availability of the documentation and support
  - Licenses and costs

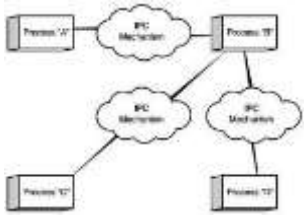
Startup and shutdown of components

It is necessary to determine how the various modules are instantiated and any dependencies on system components:

– A complex software that requires more modules (processes, drivers, etc.) usually endows a program supervisor who is responsible for

- Instantiate components in the right order
- Verify its execution and take action if an abnormality occurs
- Support the upgrade and redundancy if required
- Implement graceful termination of the components in the event of a shutdown request

Designing the interaction between processes

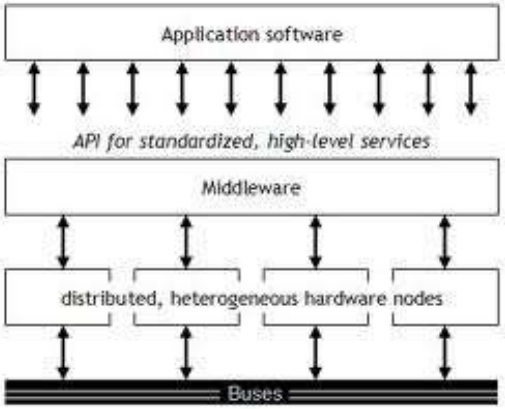


- In other words an IPC mechanism
  - Complex projects need to define and implement a communication mechanism
  - It usually has desirable qualities, such as:
    - Capability to encode information ensuring the independence from the HW architecture
    - Working also across a network
    - Based on standard protocols
    - Respecting of requirements of performance and resilience
    - Debug-able using standard tools (such as protocol analyzers)

Defining the iterations with O/S, 3PP libraries and modules

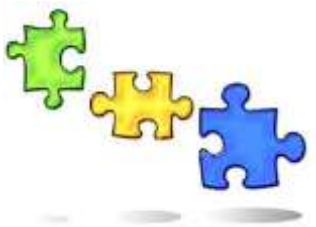
- The use of one or more layers of decoupling (middleware, adapters, proxy / stub, etc.)
- The use of facilities or 3PP libraries also defining the criteria for linking static or dynamic

- Any iterations with complex systems such as for example
  - DBMS
  - Graphics subsystems
  - Firewall, networking, etc.



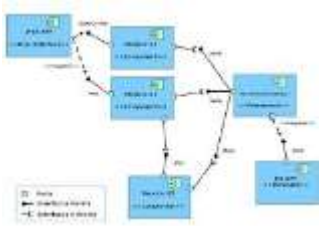
Identification of all the components to include

- Drivers and modules in kernel space
- Services (user processes that do not interact directly with the UI)
- Running applications and utilities
- Libraries (static and dynamic)
- User Interfaces:
  - CLI for monitoring and debugging
  - GUI
  - Based on communication interface
- Logger (to track the behavior of various modules)



Component design

- The design depends on the type of component, but in general it can be summarized in terms of
  - Context (or threads) of execution
  - Synchronization mechanisms
  - Iteration with other system component
  - Public or private programming interfaces
  - Definition of data structures (entities and relationships, class-diagram)
  - Use-case and BUT applicable to the component or parts of it
  - Simulation, debugging and tracing
  - Configuration



Component implementation

- This phase is closely related to the type of component, but a good implementation generally has some common characteristics:
  - is documented and written in a clear and legible way
  - is modular and testable through BUT or other simulation mechanisms
  - Respects coding style defined for the project
  - Respects the interfaces agreed in the design
  - is geared toward choices that enable compliance with the requirements of performance and resiliency required
  - is versioned in a manner consistent

Testing software

The test is divided into:

- Basic Unit Test
  - Made in the development phase and to ensure non regression due to changes: sub-parts not further decomposable are tested (eg., the implementation of a particular function or class)
- Testing the isolated system
  - Similar to the BUT but done on all the component through an appropriate interface
- Testing of the complete system, including
  - Validation tests
  - Automatic regression tests

Deployment

- The software must be distributed to be easily installed:
  - Must be distributed in a format that allows the user to install in a practical and safe way
  - Upgrade or uninstall process should be designed and implemented
    - Advanced tools for creating installation packages for desktop systems could be used

Support

- Support services attempt to help the user solve specific problems
  - May be delivered by email or on a website or a tool where users can log a call/incident
- Bug-fixing support could be provided via upgrade packages delivered on a website


That's All
Folks