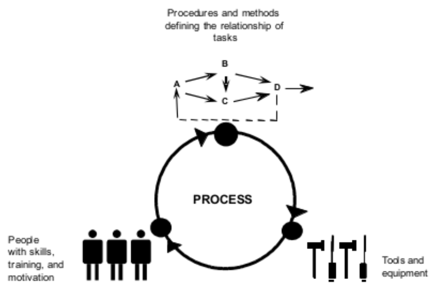
Software Engineering notes

Matteo Salvino

Contents

1 Capability Maturity Model Integration

Now days companies want to deliver products and services better, faster and cheaper. At the same time, in the high-technology environment of the twenty-first century, nearly all organizations have found themselves building increasingly complex products and services. It's unusual today for a single organization to develop all the components that compose a complex product or service. More commonly, some components are built in-house and some are acquired; then all the components are integrated into the final product or service. Organizations must be able to manage and control this complex development and maintenance process. The problems these organizations address today involve enterprise-wide solutions that require an integrated approach. Effective management of organizational assets is critical to business success. In other words, these organizations are product and service developers that need a way to manage their development activities as part of achieving their business objectives. In the current marketplace, maturity models, standard, methodologies and guidelines exist that can help an organization to improve the way it does business. However, most available improvement approaches focus on a specific part of the business and do not take a systematic approach to the problems that most organizations are facing. So, focusing on improving one particular business area, these models are hindered by barriers that exist in organizations. **CMMI** for development (CMMI-DEV) provides an opportunity to avoid or eliminate these barriers. It consists of best practices that address development activities applied to products and services. It address practices that cover the product's lifecycle from conception through delivery and maintenance. The goal is to build and maintain the total product. CMMI-DEV contains 22 process areas. Of those process areas, 16 are core process areas, 1 is a shared process area and 5 are development specific process areas. The Software Engineering Institute (SEI) in its research, has found some key points on which organizations can focus on to improve its business.


Procedures and methods defining the relationship of tasks
A
B
C
D
PROCESS
People with skills, training, and motivation
Tools and equipment

In particular these key points are : people, procedures and methods, and tools and equipment. These points are kept together by the processes used in our organization. They allow us to address scalability and provide a way to incorporate knowledge of how to do things better. Processes allow us to leverage our resources and to examine business trends. We are focusing on methodologies, not because people and technologies aren't important, but since the technology changes at an incredible speed and people can works for different companies, in this way we can provides the infrastructure and stability necessary to deal with an ever-changing world and to maximize the productivity of people and the use of technology to be competitive. The SEI has taken the process management premise "the quality of a system of product is highly influenced by the quality of the process used to develop and maintain it" and defined CMMs that embody this premise. A CMM, including CMMI, is a simplified representation of the world, which contains the essential elements of effective processes. CMMs focus on improving processes in an organization, describing an evolutionary path from an immature process to disciplined mature process with improved quality and effectiveness. CMMI can be used in process improvements also as a framework, which provides the structure needed to produce CMMI models, training and evaluation components. To allow the use of multiple models withing the CMMI framework, model components are classified as either common to all CMMI models or applicable to a specific model. The common material is called **CMMI Model Foundation** (CMF). The components of the CMF are part of every model generated from the CMMI framework. Those components are combined with material applicable to an area of interest to produce a model. A **constellation** is defined as a collection of CMMI components that are used to construct models, training materials and evaluate related documents for an area of interest. CMMI-DEV previously introduced is the development constellation for a model. All CMMI models contains multiple **Process Areas** (PAs). A process area is a cluster of related practices in an area that, when implemented collectively, satisfies a set of goals considered important from making improvement in that area. Some of these area are : Causal Analysis and Resolution (CAR), Configuration Management (CM), etc. We can define two type of goal :

- **Generic** : it's called generic because the same goal statement applies to multiple process areas. A generic goal describes the characteristics that must be present to institutionalize processes that implement a process area. It's a required model component and is used in evaluation to determine whether a process area is satisfied or not. The istitutionalization is an important concept in process improvement, since it implies that the process in ingrained in the way the work is performed and there is a commitment and consistency to performing the process. The progress of process institutionalization is characterized by the type of the generic goal :

- **GG1 Performed process** : a performed process is a process that accomplishes the work necessary to satisfy the specific goals of a process area.
- **GG2 Managed process** : a managed process is a performed process that is planned and executed in accordance with some policy. It employs skilled people having adequate resources to produce controlled outputs. It's controlled, monitored, reviewed and evaluated for adherence to its process description. The process can be instantiated by a project, group or organizational function. Management of the process is concerned with institutionalization and the achievement of other specific objectives established for the process, such as cost, schedule and quality objectives. The control provided by a managed process helps to ensure that the established process is retained during times of stress. The requirements and objectives for the process are established by the organization. The status of the work products and services are visible to management at defined points. Commitments are established among those who perform the work and the relevant stakeholders and are revised as necessary. Work products are reviewed with relevant stakeholders and are controlled. A critical distinction between a performed process and a managed process is the extent to which the process is managed. A managed process is planned and its execution is managed against the plan. Corrective actions are taken when the actual results and execution deviate significantly from the plan. A managed process achieves the objectives of the plan and is institutionalized for consistent execution.
- **GG3 Defined process** : a defined process is a managed process that is tailored from the organization's set of standard processes according to the organization's tailoring guidelines. It has a maintained process description and contributes process related experiences to the organizational process assets. Organizational process assets are artifacts that relate to describing, implementing and improving processes. These artifacts are assets because

they are developed or acquired to meet the business objectives of the organization and they represent investments by the organization that are expected to provide current and future business value. The organization's set of standard processes, which are the basis of the defined process, are established and improved over time. Standard processes describe the fundamental process elements that are expected in the defined processes. Standard processes also describe the relationships among these process elements. A project's defined process provides a basis for planning, performing and improving the project's task and activities. A critical distinction between a managed process and a defined process is the scope of application of the process descriptions, standards and procedures. For a managed process, the process descriptions, standards and procedures are applicable to a a particular project, group or organizational function. As a result, the managed processes of two projects in one organization can be different. Another critical distinction is that a defined process is described in more detail and is performed more rigorously than a managed process. Finally, management of the defined process is based on the additional insight provided by an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.

- **Specific** : a specific goal describe the unique characteristics that must be present to satisfy the process area. It's a required model component and is used in evaluation to determine whether a process area is satisfied or not.

1.1 Understanding levels

Levels are used in CMMI-DEV to describe an evolutionary path recommended for an organization that wants to improve the processes it uses to develop products or services. CMMI supports two improvement paths using levels. One path enables organizations to incrementally improve processes corresponding to an individual process area selected by the organization. The other path enables organizations to improve a set of related processes by incrementally addressing successive sets of process areas. These two improvement paths are associated with the two types of levels : **capability levels** and **maturity levels**. These levels correspond to two approaches to process improvement called **representations**. In turn, the two representations are called **continuous** and **staged**. Both representations provide ways to improve our processes to achieve business objective and use the same model components. The continuous representation is concerned with selecting both a particular process area to improve and the desired capability level for that process area. The staged representation is concerned with selecting multiple process areas to improve within a maturity level. Both capability levels and maturity levels provide a way to improve the processes of an organization and measure how well organizations can and do improve their processes. However, the associated approach to process improvement is different.

1.1.1 Capability levels (Continuous representation)

To support those who use the continuous representation, all CMMI models reflect capability levels in their design and content. The four capability levels, each of them is a layer for the process improvement, are designated by the numbers 0 through 3 :

- **0. Incomplete** : an incomplete process is a process that either is not performed or is partially performed. One or more of the specific goals of the process area are not satisfied and no generic goals exist for this level since there is no reason to institutionalize a partially performed process.
- **1. Performed** : a performed process is a process that accomplish the needed work to produce work products; the specific goals of the process area are satisfied.
- **2. Managed** : a managed process is a performed process that is planned and executed in accordance with some policy; It employs skilled people having adequate resource to produce controlled outputs; it involves relevant stakeholders; It's monitored, controlled, reviewed and evaluated for adherence to its process description. The process discipline reflected by capability level 2 helps to ensure that existing practices are retained during times of stress.
- **3. Defined** : a defined process is a managed process that is tailored from the organization's set of standard processes according to the organization's tailoring guidelines and contributes work products, measures, and other process improvement information to the organizational process assets.

1.1.2 Maturity levels (Staged representation)

To support those who use the staged representation, all CMMI models reflect maturity levels in their design and content. A maturity level is a defined evolutionary plateau for organizational process improvement. Each maturity level matures an important subset of the organization's processes, preparing it to move to the next maturity level. The maturity levels are measured by the achievement of the specific and generic goals associated with each predefined set of process areas. The five maturity levels are designated by the numbers 1 through 5 :

- **1. Initial** : in this layer processes are usually ad hoc and chaotic. In spite of this chaos, maturity level 1 organizations often produce products and services that work, but they frequently exceed the budget and schedule documented in their plans. These organizations are characterized by a tendency to overcommit, abandon their processes in a time of crisis, and be unable to repeat their successes.
- **2. Managed** : as before.
- **3. Defined** : as before.
- **4. Quantitatively managed** : it's a defined process that is controlled using statistical and other quantitative techniques. Quantitative objectives for quality and process performance are established and used as criteria in managing the process. Quality and process performance is understood in statistical terms and is managed throughout the life of the process.
- **5. Optimizing** : it's a quantitatively managed process that is improved based on understanding of the common causes of variation inherent in the process. The focus of an optimizing process is on continually improving the range of process performance through both incremental and innovative improvements.

1.2 ISO family

ISO 12207 defines and structures all activities involved in the software development process. Its main goal is to provide a common language to involved stakeholders. It's based on a functional approach : a set of coordinated activities transforming an input in an output. It's based on two basic principles :

- **Modularity** : it means processes with minimum coupling and maximum cohesion.
- **Responsibility** : it means to establish a responsibility for each process, in order to facilitate the application of the standards in a project where there are many people involved.

ISO 9000 is maintained by ISO and is administered by accreditation and certification bodies. This family of ISO addresses ”Quality management”. Its fundamental building blocks are :

- **Quality management system** : it deals with general and documentation requirements that are the foundation of the management system. In particular, the previous requirements can be explained in details :
  - **General** : they are general requirements like how the processes of the management system interact to each other or how you will measure and monitor the processes.
  - **For documentation** : they are requirement focused on the documentation. They can require what documentation is needed to operate the system effectively or how it should be controlled.
- **Management responsibility** : it manage high level responsibilities like set policies and objectives or plan how the objectives will be met.
- **Resource management** : it deals with the people and physical resources needed to carry out the process. People should be competent to carry out their task and physical resources and work environment need to be capable of ensuring that the customer’s requirements are satisfied.
- **Product-service realization** : it deals with the processes necessary to produce the product or to provide the service.
- **Measurement, analysis, and improvement** : it deals with measurements to enable the system to be monitored. For example we can measure if the processes are effective or if the product really satisfy customer’s requirements.

ISO doesn’t itself certify organizations. There are accreditation bodies that authorize certification bodies. Organizations can apply for ISO 9001 compliance certification to a certification body. The various accreditation bodies have mutual agreements with each other to ensure that certificates issued by one of the Accredited Certification Bodies (CB) are accepted worldwide. An ISO certificate is not a once-and-for-all award, but must be renewed at regular intervals recommended by the certification body, usually around three years. **Quality requirements** are a set of process requirements and resources that constitute the Quality Manual (QM) of the organization. This latter specifies the organization's quality policy regardless specific commitments and customers. It's adapted to specific projects, generating several Quality Policies (QP). The ISO 9001 certification required that processes are described in the two previous specific documents (QM and QP).

2 Software Development Process Models

Software products are not tangible. In order to manage a software project the project manager needs special methods. So, the monitoring process in based on the explicit definition of activities to be performed and documents to be produced. These documents allow us to monitor the progress of the process and give us an idea of its quality. Software Development Process Models and their instances, differ from each other for the required activities and for the produced documents. Now, we will see several software process models, discussing their pros and cons.

2.1 Waterfall model

A Waterfall model is a typical process in which there are separate and distinct phases of specification and development. So, first we describe all the specifications of the software and then we have a subsequent phase that take care about the development. Its main phases are :

- **Requirements analysis and definition**
- **System and software design**
- **Implementation and unit testing**
- **Integration and system testing**
- **Operation and maintenance**.

Each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. One of its main drawback is the difficulty of allows changes after the process in started. Another drawback is that the end users doesn't have a vision of the overall system (uncertainty). So, they have to wait till a working version of the system is available. In turn, programmers have to wait the analysis phase before starting their job. The solution is to use an iterative approach.

2.2 Process iteration

The system requirements always evolve during the project development, so process iteration where earlier stages are reworked is always part of the process for large systems. Iteration can be applied to any of the generic process models. Typically, an iteration follows one of the following approaches :

- **Incremental delivery** : we start building a small system (prototype) and next we enlarge it (incremental way). A **prototypal model** is constituted by a customer interaction in order to obtain customer's requirements, then we build a prototype also called **mock-up** (when we build a system with feels and looks similar to the final system, but without working software behind), and then we present it to the user, in order that he is able to test the prototype and say "Ok, i'm satisfied", or viceversa. If the answer is positive, then we can start to implement the functionality behind the mockup. The **incremental model** is formed by iterations which are constituted by analysis, design, implementation and test phases. The result of a generic iteration i typically is the system with version i. It's very similar to the prototypal model but in this case the intermediate version are full working and it allows for a more accurate design. Its main drawback is that if we figure out a wrong functionality in a specific system version, we have to throw away a lot of job previously done, whereas in the first model we will build only the mock-up system. In the incremental development we define the requirements, then we assign each of them to a specific system release version, design the whole system architecture, then develop, validate and integrate the system increment, and finally validate the system. If this isn't the last iteration, then we continue to increment the functionality of our system, otherwise we have built the final system. In other words, delivering part of the required functionality. Then the user requirements are prioritized and the highest priority requirements are included in early increments (early versions). Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve. The trade off is the length of these iterations, the bigger the interval is then we move towards the waterfall model, the smaller the interval is then will be more difficult to build a very good software architecture. Its advantages are the following : the customer can see at each iteration an increment of the system functionalities, early increments act as a prototype to help asking requirement for future iterations, lower risk of overall project failure and the highest priority system requirements tends to receive the most testing.

- **Spiral development** : in spiral development the process is represented as spiral (rather than a sequence) of activities. Each loop in the spiral represents a phase in the process. There not exist fixed phases such as specification or design (the loops in the spiral are chosen depending on what is required). The risks are explicitly evaluated and resolved throughout the process. In particular, the spiral model is constituted by the following sectors :
  - **Objective setting** : they are specific objectives for the current phase.
  - **Risk assessment and reduction** : the risks are evaluated and activities punt in place to reduce the key risks.
  - **Development and validation** : in this sector we choose a development model for the system.
  - **Planning** : the project is reviewed and the next phase of the spiral is planned.

2.3 Formal methods

Formal methods are formalisms based on logic or algebra for requirement specification, development and test. They don't use the natural language, because it's very ambiguous, but tends to write the specification of the software in some formal languages like Z, Z++, etc.

2.4 Extreme programming

It's a part of the Agile model family. It's an approach based on the development and delivery of very small increments of functionality. It relies on constant code improvement, use involvement in the development team and pairwise programming.

2.5 Core process activities

Now, lets quickly review the main activities involved in the development process :

- **Software specification** : it's the process of establishing what services are required and the constraints on the system operations and development. These requirements can be functional or non-functional (they regards about system quality). This process is also called requirements engineering process. First of all it perform a feasibility study to understand if building the system is feasible. Typically this phase produce a document called feasibility report. Subsequently, the requirements are defined, analyzed and validated. The results of these phases constitute the so called requirement document.

- **Software design and implementation** : it's the process of converting the system specification into an executable system. The software design phase is a process that in which we design the software structure (architectural, component, data structures, etc.) that fulfill the specifications. In the implementation phase we translate the previous structure into an executable program. In this phase our goal is also to remove as many errors as possible from the program generated, typically using a program testing. These two phases are closely related and may be interleaved.

- **Software validation** : the verification and validation is intended to show that the system is conform to its specifications (verification) and meets the customer's requirements (validation). Typically the system is tested over test cases that are derived from the specification of the real data to be processed by the system. We can use several types of testing :

  - **Unit test** : individual components are tested independently.
  - **System test** : test the whole system.
  - **Acceptance test** : testing with customer data to check that the system meets the customer's needs.

- **Software evolution** : software is very flexible and can change over time. A change of the requirements must be reflected also in the software.

3 Scrum

Scrum is an agile process that allows us to focus on delivering the highest business value in the shortest time. It allows us to rapidly and repeatedly inspect actual working software (every two week to one month). The business set the priorities. Team self-organize to determine the best way to deliver the highest priority features. Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance it for another sprint. The requirements of the customer are captured as items in a list of ”product backlog”. The core values of Scrum are the following :

- **Commitment** : teams commit to their goals for the sprint, product owners commit to ordering the product backlog and ScrumMasters commit to removing any obstacles along the way in order to simplify the flow of product development. The Scrum team should do whatever is necessary in order to meet their goals, and it’s important that they are empowered to do so.

- **Focus** : for a team to be able to complete its work, its members must be allowed to focus. The ScrumMaster doesn’t allow changes in the sprint’s commitment so that the team may keep its focus. When a team gives its full attention to the problem, its work is much more productive, predictable and fulfilling.

- **Openness** : as Scrum uses empirical process control to make progress through a project, it’s essential that the results and experience of an experiment (i.e. a sprint) are visible. Once visibility exists, inspection and adaption can occur.

- **Respect** : in order to be its best, a team’s members need to respect for each other and, the knowledge that each brings to the table, experiences, working styles and personalities. Respect doesn’t come for free, it’s earned. Scrum team members should be dedicated, cross-functional, empowered and self-organizing.

- **Courage** : it takes buckets of courage for a ScrumMaster to apply Scrum the way it was intended. One of the primary responsibilities of the ScrumMaster is to help the organization identify its weaknesses so that it may improve. This takes courage. Sometimes, a team has to push back on the product owner when asked to take on too much during a sprint. It takes courage to say no to that sort of pressure.

A product owner must have courage when communicating with other stakeholders about the reality of a project.

3.1 Roles

Scrum team The Scrum team includes the product owner, ScrumMaster and the team members, whereas the Scrum delivery team is a subset made of only the technical team members. The whole Scrum team huddles around a problem (i.e. a requirement from the Product Backlog) and innovates solutions. Scrum teams should be five to nine team members, dedicated to the life of the project, cross-functional, empowered and self-organizing. Scrum teams plan, estimate and commit to their work, rather than a manager performing these activities for them. The end goal of the team is to deliver a potentially shippable product increment that meets an agreed-upon Definition of Done each and every sprint.

Product owner The product owner is responsible for the product's success. In other words, while the team is responsible for delivering a quality solution, the product owner is responsible for knowing his market and user needs well enough to guide the team towards a marketable release sprint after sprint. In a project there should be one and only one product owner who makes final decisions about the direction of the product and the order in which features should be developed. The product owner, since he is represent the "what" and "why" of the system, should be available to the team to have regular dialog about the requirements in the product backlog; additionally, the product owner must make the product vision clear to everyone on the team and regularly maintain the product backlog in keeping with the product vision. The product owner always keeps the next set of product backlog items in a ready state so that the team always has work in the queue for the next sprint.

ScrumMaster The ScrumMaster safeguards the process. He/she understands the reasons behind and for an empirical process, and does his or her best to keep product development flowing as smoothly as possible. This leader protects team members from interruptions in order to keep them focused on their sprint commitments. The ScrumMaster also facilitate all Scrum meetings, ensuring that everyone on the team understands the goals and that they share a commitment together as a true team and not just as a collection of individuals. She/he removes obstacles that prevent a steady flow of high-value features.

3.2 Ceremonies

A sprint is an iteration defined by a fixed start and end data; it's kicked off by sprint planning and concluded by the sprint review and retrospective. The team meets daily, in a daily scrum meeting, to make their work visible to each other and synchronize based on what they've learned. Lets discuss in details these phases one at a time.

Sprint planning During sprint planning the product owner and the team discuss the highest priority items in the product backlog and brainstorm a plan to implement those items. The set of chosen product backlog items and their subsequent tasks collectively is referred to as the team's sprint backlog. The sprint planning meeting is time-boxed to eight hours for a 30-day sprint, reduced proportionally for shorter sprints. The meeting is constituted by two parts : the first one is driven by the product owner who presents the most important product backlog items (with the support of drawings, mockups, etc.) and clarifies question from the development team about what he/she wants and why he/she wants it. The second part is driven by the Scrum delivery team who work together to brainstorm approach and eventually agree on a plan. It's at the start of this second part that the sprint actually begins. Of course, teams are always searching for ways to make planning faster and more efficient. The result of sprint planning is a sprint backlog that is comprised of selected product backlog items for the sprint, along with the correlating tasks identified by the team in the second part of sprint planning.

Sprint review The sprint review provides the opportunity for stakeholders to give feedback about the emerging product in a collaborative setting. In this meeting, the team, product owner, ScrumMaster and any interested stakeholders meet to review and talk about how the product is shaping up, which features may need to change and perhaps discuss new ideas to add to the product backlog. It's common for a ScrumMaster to summarize the event of the sprint, any major obstacles that the team ran into, and so on, and of course the team should always demo what they've accomplished by the sprint's end. This meeting is time-boxed to four hours for a 30-day sprint.

Sprint retrospective During the final spring meeting, the sprint retrospective, team members discuss events of the sprint, identify what worked well for them, what didn't work so well and take on action items for any changes that they would like to make for the next sprint. The ScrumMaster will take on any actions that the team doesn't feel it can handle. The Scrum-Master reports progress to the team regarding these obstacles in subsequent sprints. This meeting is time-boxed to three hours.

Daily scrum meeting In the daily scrum meeting team members make their progress visible so that they can inspect and adapt toward meeting their goals. The meeting is held at the same time and in the same place, decided upon by the team. Even though a team makes its best attempt at planning for a sprint, things can change in flight. In this 15-minute meeting, team members discuss what they did since yesterday's meeting, what they plan to do by tomorrow's meeting, and to mention any obstacles that may be in their way. The ScrumMaster record any obstacles that the team members feel they cannot fix for themselves and will attempt to remove them after the meeting. The scrum delivery team members, product owner and ScrumMaster are participants in the meeting. Anyone else is welcome to attend but only as observers.

3.3 Artifacts

Product backlog The product backlog is the product owner's "wish list". Anything and everything that they think they might want in the product goes in this list. The product owner maintains the product backlog, although other stakeholders should have visibility of and the ability to suggest new items for the list. The product owner prioritizes the product backlog, listing the most important or most valuable items first. Once a team selects items for a sprint, those items and their priorities are locked; however, priorities and details for any not-started work may change at any time. Through this mechanism, teams are able to focus on this sprint's work while the product owner retains maximum flexibility in ordering the next sprint's work.

Sprint backlog Owned by the team, the sprint backlog reflects the product backlog items that the team committed to in sprint planning, as well as the subsequent tasks and reminders. Team members update it every day to reflect how many hours remain on his/her task; team members may also remove tasks, add tasks or change tasks as the sprint is started.

Product increment The product increment is a set of features, user stories or other deliverables completed by the team in the sprint. It should be potentially shippable (i.e. it must have enough quality to give it to the users). The product owner is responsible for accepting the product increment during each sprint, according to the agreed-upon Definition of Done and acceptance criteria for each sprint deliverable. Without a product increment, the product owner and other stakeholders have no way to inspect and adapt the product. A team must keep its progress visible at all times. It will create many additional artifacts in order to ensure visibility. Some common visibility tools are the release and sprint burndown charts. A burndown chart display of what work has been completed and what is left to complete. It is provided for each developer or work item. It's updated every day and make best guess about hours/points completed each day. A possible variation of this chart is called release burndown chart, which display how much work remain in the release backlog at the end each sprint (it shows overall progress).

4 Layers vs Tiers

Typically an information system is constituted by three layers : presentation, application logic and resource management layer. The **presentation layer** is devoted to present the system's user interface to clients. The **application logic** determines what the system actually does. It takes care of enforcing business processes. The application logic can take many forms : programs, constraints, etc. The **resource manager** deals with the organization (storage, indexing and retrieval) of the data necessary to support the application logic. This is typically a database but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence. A client is any user or program that wants to perform an operation over the information system. Clients are independent of each other : one could have several presentation layers depending on what each client wants to do. One can take advantage of the computing power at the client machine to have more sophisticated presentation layers. This also saves computer resources at the server machine. It introduces the concept of API, an interface to invoke the system from the outside. It also allow designers to think about federating the systems into a single system. The resource manager only sees one client : the application logic. This improve a lot the performance since there aren't client connections/session to maintain. A **tier** is a physical level. How can we divide logical layers over physical objects ? The first architecture developed is the so called **1-tier architecture**, in which all layers are running on the same machine (i.e. a mainframe). The second approach is called **2-tier architecture**, in which we can have one machine (server) that contains the application logic and resource management and another one (client) that contains the presentation layer. This approach is also called client-server architecture. The evolution of this concept is to introduce micro services, and using a **3-tier architecture**, in which the presentation layer runs on the client machine, the application logic layer on a middle machine and the resource management layer on a dedicated machine. The problem is that now, we have more connections respect to the previous architectures. In fact, the middle's developers needs to write the code for receiving requests from the client and from the server. So, the people realize that needed something more, a specific technologies called **middleware**, that could make the development of the code as much simple as if the code is still running in the mainframe (on the same machine). Once you know how to split the software in three layers, you can split each of them in more layers on its own machine, in order to make load balance. The result of this idea is called **N-tier architecture**.

4.1 Middleware

A middleware can serves the requests in two ways :

- **Synchronous** : traditionally, distributed applications use blocking calls, in which the client sends a request to a service and waits for a response of the service to come back before continuing doing its work. This type of interaction require both parties to be online : the client make the request, the receiver gets the request, processes it and sends a response to the caller, the client receives the response. Due it synchronizes client and server, this approach has several disadvantages :

  - **connection overhead** : synchronous invocations require to maintain a session between the caller and the receiver. Maintaining a session is expensive and consumes CPU resources. There is also a limit on the number of session that can be active at the same time. For this reason, client/server systems often use connection pooling (pooling of connections, with a thread associate at each of them and allocate one of them when needed) to optimize resource utilization. Synchronous interaction requires a context for each call and a context management system for all incoming calls. The context needs to be passed around with each call as it identifies the session, the client and the nature of the interaction.

  - **higher probability of failures** : if the client or the server for some reason fail, the context is lost and resynchronization is difficult. Finding out when the failure took place may not be easy.

  To the previous problems there are two solutions :

  - **Enhanced support** : we can use transactional interaction to enforce exactly once execution semantics and enable more complex interactions with some executions guarantees. Another approach is service replication and load balancing, in order to prevent the service from becoming unavailable when there is a failure.
  - **Asynchronous interaction** : see below.

- **Asynchronous** : using asynchronous interaction, the caller sends a message that gets stored somewhere until the receiver reads it and sends a response. The response is sent in a similar manner as before. This type of interaction can take place in two forms :
  - **non-blocking invocation** : a service invocation but the call returns immediately without waiting for a response, similar to batch jobs.
  - **persistent queue** : the call and the response are actually persistently stored until they are accessed by the client and the server.

A middleware can be implemented with two approaches :

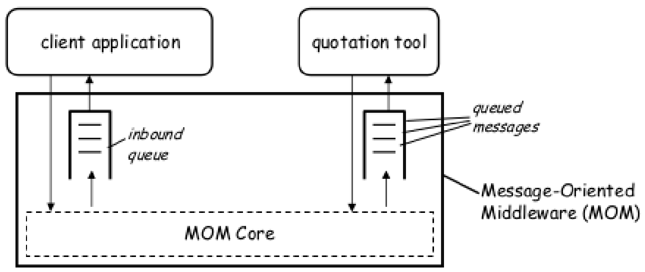
- **Programming abstraction** : it's an approach intended to hide low level details of hardware, network and distribution. Middleware is primarily a set of programming abstractions developed to facilitate the development of complex distributed systems; to understands a middleware platform one needs to understand its programming model. From this latter can be determined its limitations, general performance and applicability of a given type of middleware. Furthermore, this model also determines how the platform will evolve.

- **Infrastructure** : as the programming abstraction reach higher and higher levels, also the infrastructure that implements the abstractions must grow accordingly. Additional functionality is always implemented through additional software layer. The additional software layers increase the size and the complexity of the infrastructure necessary to use the new abstractions. The infrastructure is also intended to support additional functionality that makes development, maintenance and monitoring easier and less costly. It takes care of all non-functional properties typically ignored by programming models and languages such as performance, maintenance, resource management, etc.

Typically a middleware is based on RPC, in which the client invoke a local procedure talking with its stub (client stub), which is able to communicate with the remote object via a communication module. A request is created putting the client parameters, serialized (marshal operation) and send to the server stub. The server dispatcher deliver the serialized request to its stub, which deserialize (unmarshal operation) it, extract the parameters and finally call the server procedure. When there are more entities interacting with each other, RPC treats the calls as independent of each other. The main drawback of RPC is that recovering from partial system failures may be very complex. So, a programmer instead to implement the whole infrastructure for a distributed application can use an RPC system, which hides distribution behind procedure calls, provides an interface definition language (IDL) to describe the services, generates all the additional code necessary to make a procedure call remote and to deal with all the communication aspects and provides a binder in case it has a distributed name and directory service system. The solution to this limitation is to make RPC calls transactional, i.e. instead of providing plain RPC, the system should provide TRPC. It's based on the same concepts of RPC but, use additional language constructs and runtime support to bundle several RPC calls into an atomic unit.

4.1.1 Message Oriented Middleware

MOM is software or hardware infrastructure supporting sending and receiving messages between distributed systems, with the following structure :



The participants of the communication are divided into two groups :

- **Publisher** : the entity that send messages
- **Subscriber** : the entity that express interest in certain categories of messages.

The main disadvantage of this approach is that require an extra component in the architecture called **event service** that receives messages from publisher and cross-check them with the interests of subscribers. As with any system, an introduction of another component can lead to performance and reliability reduction, and can make the whole system more difficult and expensive to maintain. The information can be processed in two ways :

- **Topic-based** : the publisher is responsible for defining the topics to which subscribers can subscribe.
- **Content-based** : filters can be used for a more accurate selection of information to be received.

The publisher can specify which topics are available for a subscription (topic-based) and subscribers can specify particular attribute in order to filtering the messages of interest. The notification can be made in two ways :

- **push** : the subscribers are invoked in callback, using a reference communicated at the time of subscription.
- **pull** : the subscribers poll the event service when they need messages.

To-Do : add the pseudocode !

5 Web Services

Web services are the current evolution of middleware technology. Basically are offered in a way over Internet. In other words, a web service is a software functionality exposed over the Internet. This means that any piece of code and any application component deployed on a system can be transformed into a network-available service. The main difference between middleware and web services, is Internet, because the latter emerged at the beginning of 2000 (when vendors invented new technologies and standard). So, web services perform business functions such as :

- a self-contained business task (for example a funds deposit service)

- the whole business process (for example the automated purchasing of office supplies)
- an entire application (for example demand forecasts and stock replenishment)
- a service-enable resource (for example access to a particular back-end database containing some interesting data).

Once we have these functionalities exposed over Internet we can mix and match them to create a complete process. Until the emergence of web service technology the client and server should be on the same web platform (OS or programming language). With web service we can use whatever programming language and operating system that talks each other (i.e. platform independent). When we develop a web service we need to take into account the billing model, i.e. the mode in which the customer can pay in order to use specific functionalities of the service exposed. An Application Service Providers (ASP) is based on the idea to rent its own applications to subscribers such that :

- the whole application is developed in terms of user interface, workflow, business and data components that are all bound together to provide a working solution.
- an ASP hosts the entire application and the customer has little opportunity to customize it beyond the appearance of the user interface.
- an alternative of this is where the ASP is providing a software module that is downloaded to the customer's site on demand (situations in which the software can operate remotely via a browser).

When we have an ASP model it offers the whole application suite to the customers, which introduce the concept of software-as-a-service (i.e. pay for the application but we don't own it). This approach has several limitations such as : inability to develop high interactive applications, inability to provide complete customizable application and inability to integrate applications. With Web Services is different because we aren't paying for the whole application, but many different providers offers the functionalities that are needed for some someone to develop the software. This is the key difference between ASP and Web Service approaches.

A User story

BDD asks questions about the behavior of the application before and during development to reduce miscommunication. The application's requirements are written as **user stories**. A user story has the following structure :

- **As a** [kind of stakeholder]
- **So that** [I can achieve some goal]
- **I want to** [do some task].

The idea is to use user stories as acceptance test before the code is written. A measure of team productivity could be average number of stories / week ? We have a problem that some stories could be more difficult than others. So, a simple fix is to assign to each user story a rate on a simple integer scale : for example we could assign 1 for simple stories, 2 for medium stories and 3 fro very complex stories. In this new settings, the velocity is given by the relationship between the average number of points and the week. There are some guidelines in order to define properly a user story. One of those is, if the rate assigned to a user story is greater of equal to 5, then we could think to split this story into simpler stories. Another guideline which each user story should follows, is the SMART approach :

- **Specific and Measurable** : each scenario should be testable. It implies known good input and expected results exist. For example Given some specific starting condition, When i take specific action X, Then one or more specific event should happen.
- **Achievable** : the ideal case is to complete user stories in one iteration. In real world, this situation never occurs. If we can't deliver feature in one iteration then deliver a subset of stories.
- **Relevant** : of course, the user stories should represent an important feature. (see also 5 Whys approach). Otherwise we can delete useless stories.
- **Timeboxed** : naturally if a story exceed the time budget we need to stop it. We can divide it in simpler stories or reschedule what is left undone. This is an important characteristic, because in this way we avoid to underestimate the length of the project.

When working with customer is useful to present user stories also in graphical form via Lo-Fi UI that give to the customer a high level idea of how the application will looks. Another important aspect is to include also a storyboard, which show how the UI changes based on user actions.