Software Engineering notes

Mattia Nicolella

2018-2019

Contents

Chapter 1

Standardization of software processes

1.1 Introduction to process models

The main factors in software development are three:

- People
- Technology
- Development process

Organizations try to improve the quality of their software by improving their development processes.

Definition. *A process model is a structured collection of practices that describe the characteristics of effective processes*

Practices include into a process model are proven to be effective by experience. A process model has many uses:

- sets measurable objectives and priorities for a development process
- ensures stable, capable and measurable processes
- is a guide for improvement of project and organizational processes
- is used to diagnose and certify the state of current practices

1.2 Capability Maturity Model Integration

CMMs were initially developed for the American Department of Defense by the Software Engineering Institute in 1980, since then their popularity increased and they were added in modern CMMI.

Definition. *CMMI is a process improvement approach that provides organizations with the essential elements of effective processes.*

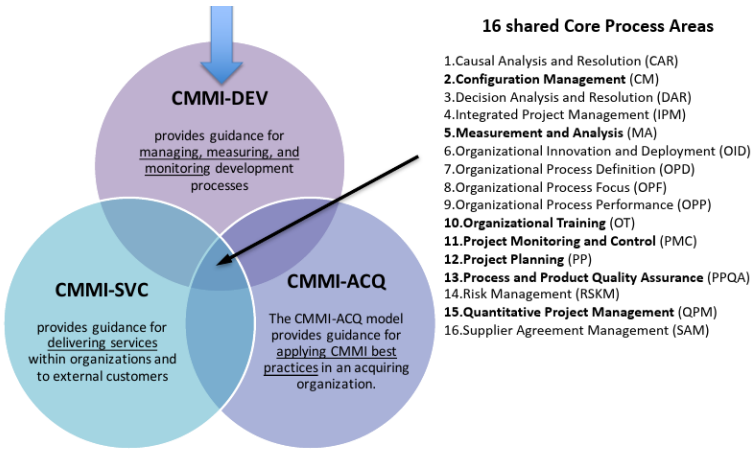


Figure 1.1: CMMI constellations

CMMI can be used either as a collection of best practices, each addressing a different area of interest, or as a framework, organized in constellations (Fig. 1.1), to prioritize and organize activities.

An Organization using CMMI has a capability level assigned, which start from 0 and can go up to 5. (Fig. 1.2)

0 - Incomplete. This is not a CMMI level since the process has not been fulfilled in at least one of its goals.

1 - Performed. The specific goals are fulfilled and the it supports the work to create the required products.

2 - Managed. The process has the basic infrastructure in place to support the process. It is planned and executed in accordance with policy; employs skilled people who have adequate resources to produce controlled outputs; involves relevant stakeholders; is monitored, controlled, and reviewed; and is 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. The managed process is tailored from the organization's set of standard processes according to the organization's tailoring guidelines, and contributes to the organizational process assets.

4 - Quantitatively Managed. Defined process that is controlled using statistical and other quantitative techniques. Quality and process performance is understood in statistical terms and is managed throughout the life of the process.

5 - Optimizing. Quantitatively managed process that is improved continually based on an understanding of the common causes of variation inherent in the process and through both incremental and innovative improvements.

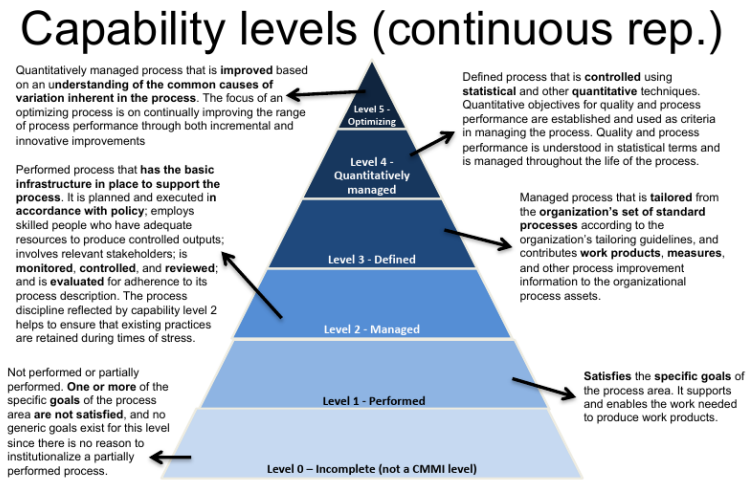


Figure 1.2: Organization capability levels

All CMMI models have multiple process area (PA) (Fig. 1.3) that have specific goals to satisfy (up to 4 different goals) and practices, there are also generic goals that are common to all the process areas.

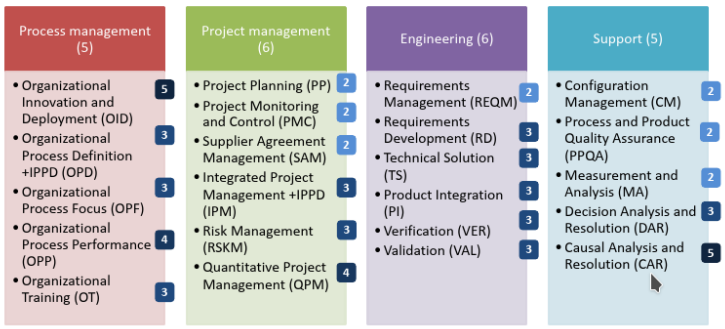


Figure 1.3: Process Areas

Each PA goals are not meant to be satisfied all at once, but each goal need to be satisfied once a determined level is reached. (Fig. 1.4)

1.3 ISO 12207 standard

Definition. *The ISO 12207 standard for sotware lifecycle processes defines and structures all activities involved in a software development process. It is based on a function approach and its main goal is to provide a common language to involved stakeholders.*

It is composed by: (Fig.1.5)

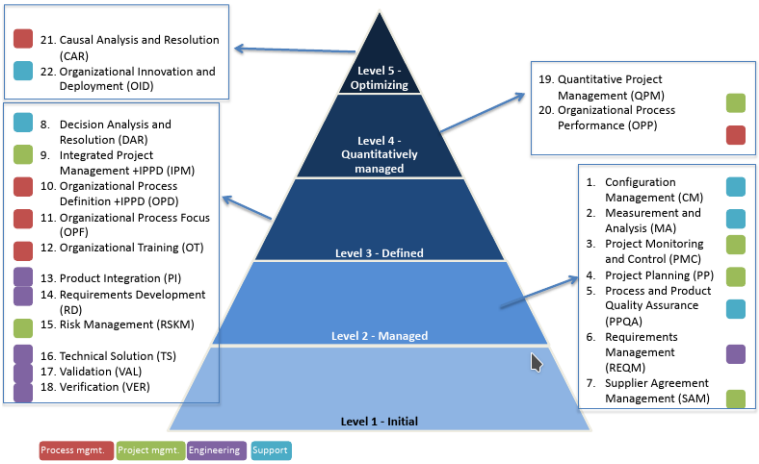


Figure 1.4: PAs in relation to the CMMI level

- 5 lifecycle processes
- 8 supporting lifecycle processes
- 4 organizational processes

Each process is specifiedin terms of activites and has a specified set of outcomes.

The ISO 12207 standard is based on two fundamental properties: **modularity** and **responsibility**. Which aim to make each module as independent as possible and to establish responsibility for each process; to facilitate operation with many processes and many legal actors involved.

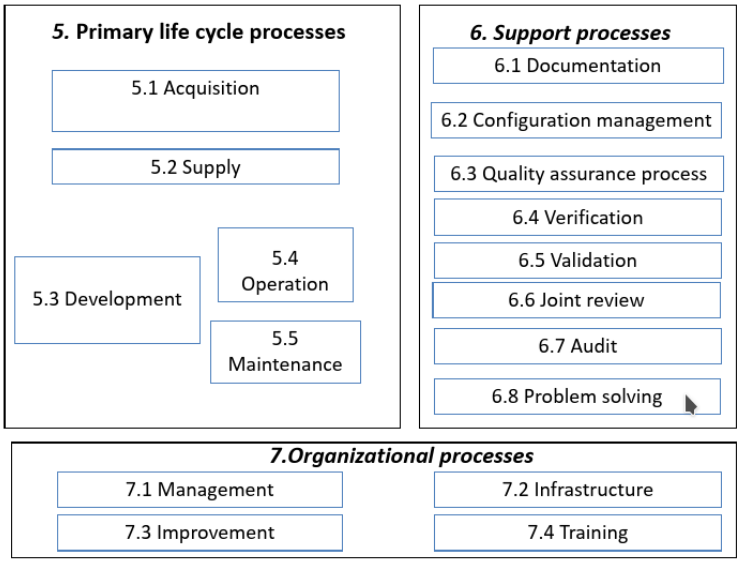


Figure 1.5: ISO 12207 Processes

1.4 ISO 9000 standards for quality management systems

The ISO 9000 family addresses "Quality management" and has several standards:

ISO 9000:2015 Fundamentals and vocabulary Description of the core language for the complete ISO 9000 family. Also describes the seven principles of quality management with tips to ensure they are reflected in the work. It also contains term and definition for other standards in the family, like ISO 9001.

ISO 9001:2015 Requirements It is intended for being used in any organization which designs, develops, manufactures, installs and/or services any product or provides any form of service. It provides requirements necessary to achieve customer satisfacton, the requirements for the continual improvement of the product/service. It is the target of the certification process.

ISO 9004:2009: Guidance for Performance Improvement Guidelines to improve and already mature system, covering continual improvement.

ISO 19011:2012 Guidance to perform internal and external audits to ISO 9001, preparing the organization to seek and external certification.

The certification for ISO 9001 gives several advantages such as control over key processes, better effciency, continual improvement, effective risk management and potential world-wide recognition; however this certification comes with several disadvantages such as the need for more documentation to be produced, is difficult to implement, it too abstract and it costly to maintain and to obtain.

The ISO 9000 has several building blocks:

Quality management system General documentation requirements that are the foundation of the management system, what documentation need to be produced and how it must be controlled. It has also some general requirements for processes, defining how they must interact with each other, what resources are needed and how to measure and monitor the processes.

Management responsibility Top management rules, which comprehend:

- knowledge of the customer requirements at a strategic level and commitment to meet these requirements
- set policies and objectives
- plan to meet the objectives
- ensuring clear internal communication
- regular reviews of the management system

Resource management Deals with people capabilities requirements and resource requirements to carry out task that will ensure customer satisfaction.

Product/service realization Deals with the process that produces the service/output.

Measurement, analysis, and improvement Measurements to take to make system measurable, which can monitor system performance with internal audits, process effectiveness and customer requirements satisfaction level to improve the system.

1.4.1 ISO certification

The certification is released by third-part certification bodies that are authorized by accreditation bodies, both of them take fees and ensure the world-wide recognition of the certification, which needs to be renewed usually every three years.

The ISO 9001 certification require two documents (Fig. 1.6) to describe each process:

Quality manual General quality policy of the organization regarding a set of requirements and resources.

Quality policy Adaptation of the quality manual to a specific project

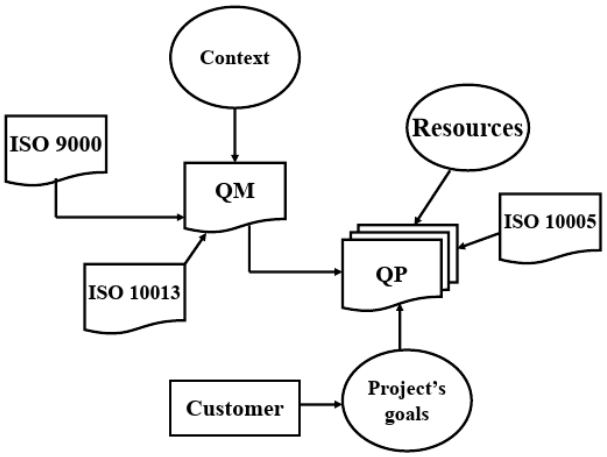


Figure 1.6: Documentation scheme

Chapter 2

Software Development Process Models

2.1 Software Development Process Modeling

Software products are not tangible, so to manage a software project there are some specific methods. Monitoring a software project is based on the definition of specific documents to be produced and the activities to be performed. These documents allow to monitor and check the evolution of the project over time, providing means to evaluate also its quality. Every model has specific criteria to produce documentation and to perform activities; in general a good compromise is to produce only the documentation that is useful for the project.

2.1.1 Waterfall model

Characteristics. In this model the specification and development phases are distinct. In detail the model is composed of 5 phases: (Fig. 2.1)

1. Requirements analysis and definition
2. System and software design
3. Implementation and unit testing
4. Integration and system testing
5. Operation and maintenance

Each of these phases has to be complete before moving to the next phase.

Drawbacks. This model does not permit changes after a phase has been completed. This causes an initial uncertainty because at the beginning of the project the vision of the overall system is not clear. The customer needs to wait until a working version of the system is ready, which happens near the end of the project. These disadvantages lead to an high probability that at the end of the project the customer satisfaction is not achieved.

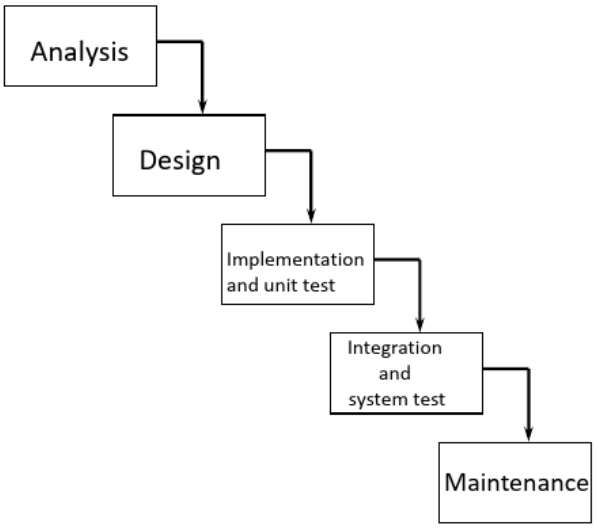


Figure 2.1: Waterfall model flow between project phases

2.1.2 Process iteration

Characteristics. In this model requirements always evolve in the course of the project so its common to have iterations where the earlier project phases are reworked, moreover these iteration can be applied to any of the generic process models. There are two related approaches: Incremental delivery and spiral development.

Incremental delivery

Has also two approaches, the prototypal model and the incremental model.

Prototypal model. This model is built on a working prototype (Fig. 2.2) which implements only some basic functions; this is used to disambiguate requirements before proceeding to implement the given requirements. A prototype is create for each set of requirements, before actually implementing them in detail.

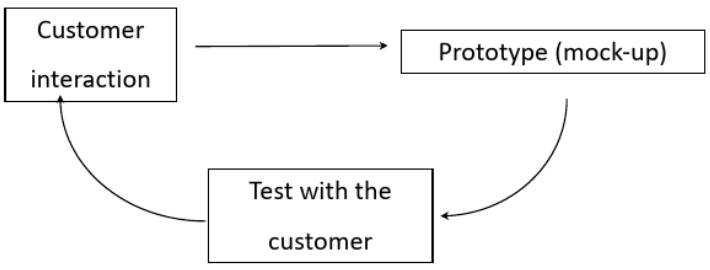


Figure 2.2: Prototypal model

Incremental model. This model is similar to the prototype model, however instead of using several prototypes it uses many incremental versions (Fig. 2.3) of the product that are fully working to disambiguate the requirements. Each version includes or modifies the functions of the previous ones, allowing a more accurate design of the whole product.

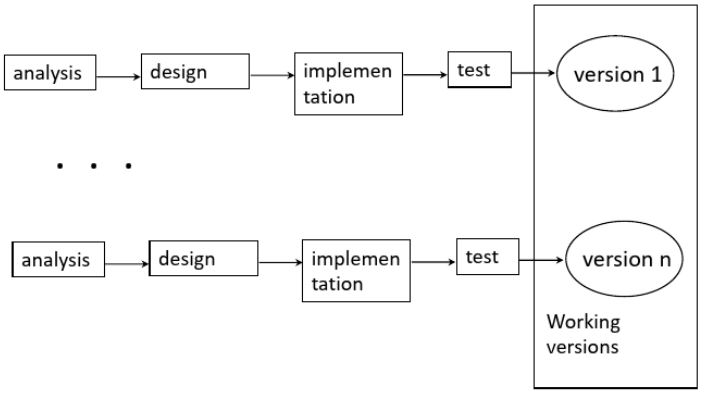


Figure 2.3: Incremental model

Incremental development. Both the previous models are part of the incremental development and delivery model (Fig. 2.4), which focuses on delivering a part of the required functionalities per cycle, prioritizing user requirements (the one with the highest priority are implemented first). Once the development of a set of requirements is started, the next requirements need to wait the completion of the current development phase to be evaluated, developed and integrated.

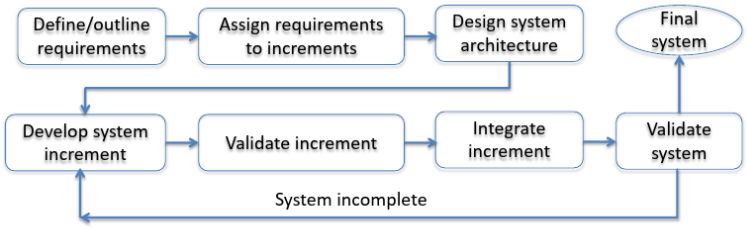


Figure 2.4: Incremental development workflow

Advantages. By developing earlier the most important requirements we guarantee that they are the most tested in the system, since every test tests the whole project, helping to delivery functionalities earlier and to increment customer value. The incremental structure of the development makes the whole process able to clarify and elicit requirements that cannot be made or specified in the early phases; this makes the overall risk of project failure lower.

Spiral model

This model represents the process as spiral (Fig. 2.5), where a phase in the process is represented by a loop in the spiral. Each loop is chosen depending on the requirements and risks are explicitly assessed ad resolved in each phase.

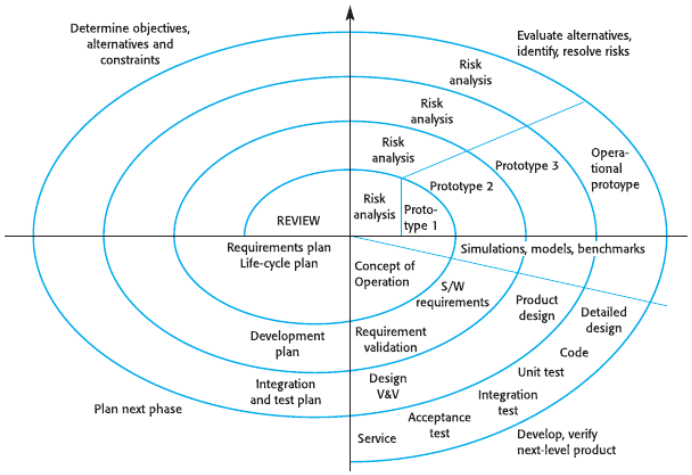


Figure 2.5: Spiral development workflow

The spiral model has several sectors which are all traversed in one loop:

Objective setting: Specific objectives for the phase are identified.

Risk assessment and reduction: Risks are assessed and activities put in place to reduce the key risks.

Development and validation: A development model for the system is chosen which can be any of the generic models.

Planning: The project is reviewed and the next phase of the spiral is planned.

2.1.3 Formal methods

Formal methods focus on removing ambiguities in the requirement specification, by using formal languages (like Z,Z++) with a specific and unambiguous syntax, allowing to avoid misunderstandings during the specification, development and test phases.

2.1.4 Extreme Programming

This model is part of the Agile model family, so like all Agile models relies on user involvement during development, constant code improvement and delivering of small increment of functionalities. This model differentiate from the other models in the Agile family because it relies on pairwise programming, a technique where programmers code in couples (one codes while the other checks errors), to reduce mistakes and produce better code.

2.2 Process activities

2.2.1 Software specification (requirements engineering)

In this phase functional and non-functional requirements are specified.

Definition. *Functional requirements are requirements in where the functionalities of the services to be implemented are specified*

Definition. *non-functional requirements are requirements where constrains on the system are specified.*

Quality requirements and performance requirements are an example of non-functional requirements.

The specified requirements are must be analyzed and validated and a feasibility study is performed, to understand if the whole project in realizable or not. (Fig. 2.6)

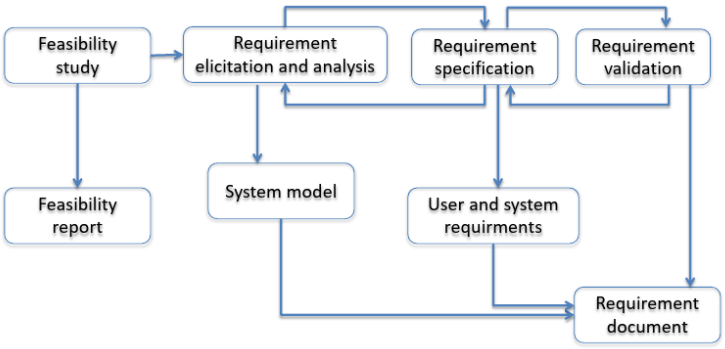


Figure 2.6: Requirement engineering process

2.2.2 Software design and implementation

In this phase a software system is designed and realized, the design and the realization phase can be interleaved.

Design phase. A software structure is designed to accomplish the specifications, in detail the design involves:

- Architectural design
- Interface design, both software interfaces and user interfaces
- Component design
- Data structure design
- Algorithm design

The design phase is documented as a set of models, mostly graphical, and the most used model to do so is the UML model.

Realization phase. The realization phase uses the results of the design phase to translate them into an executable program, removing most of the errors; programming however is creative activity and there is no generic programming process, so developers are required to test their code and remove faults.

Debugging process. In the debugging process developers locate the errors in their code, design the error repair procedure, repair the error and re-test the program, to be sure that they haven't introduced other errors and that the discovered error has been removed.

2.2.3 Software validation and verification

The validation of the system checks if it conforms to the specifications (validation) and meets the customer requirements (verification), the system is tested in with a sample of the real data it must handle.

Testing stages. In the testing phase there are several stages (Fig. 2.7), to ensure that the test is accurate:

Unit testing The single components are tested independently (functions, objects or small coherent grouping of the two).

System testing The system as a whole is tested, to understand if interaction between units can cause problems.

Acceptance testing Test with customer data to be sure that the system meets the customer needs.

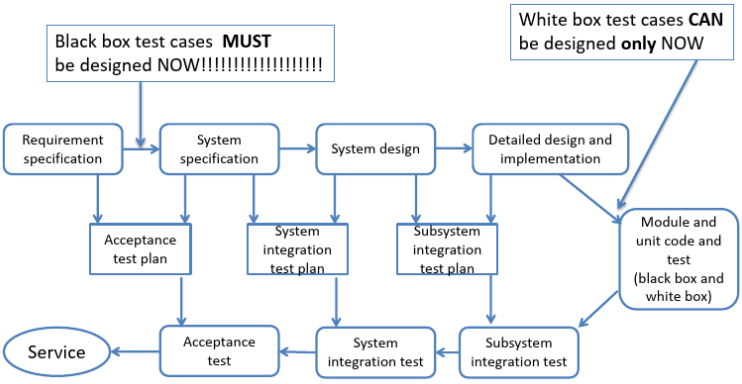


Figure 2.7: Testing workflow

2.2.4 Software evolution

Software is inherently flexible and can change because requirements can change due to changes in business circumstances. The software that supports the business must also evolve and change as the business changes. (Fig. 2.8) Although there has been a demarcation between development and evolution

(maintenance), this is increasingly irrelevant as fewer and fewer systems are completely new. Software evolution takes into account existing system (other than the one evolving) to plan the evolution of the current system.

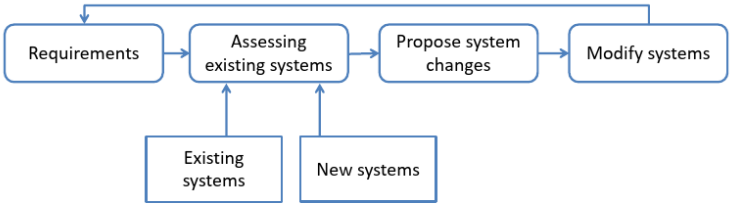


Figure 2.8: Software evolution workflow

Chapter 3

An Introduction to 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 weeks to one month). The business sets the priorities. Teams 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. This methodology is currently used by several organizations (like IBM, Microsoft and Google) to produce produce different products (ISO 9001 certified products, software for several applications etc.).

The Agile Manifesto. Scrum is part of the Agile family of processes, so it is compliant with the Agile manifesto, which has the following key concepts:

- Individual and interactions over process and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan;

Each project, however has a ”noise level” (Fig. 3.1) which depends on the level of agreement on the requirements and on the availability of the needed technologies.

Scrum as a process model has several characteristics, which are compliant with the Agile family of processes.

- Self-organizing teams
- Product progresses in a series of 2-4 week “sprints”
- Requirements are captured as items in a list of “product backlog”
- No specific engineering practices prescribed
- Uses generative rules to create an agile environment for delivering projects
- One of the “agile processes”

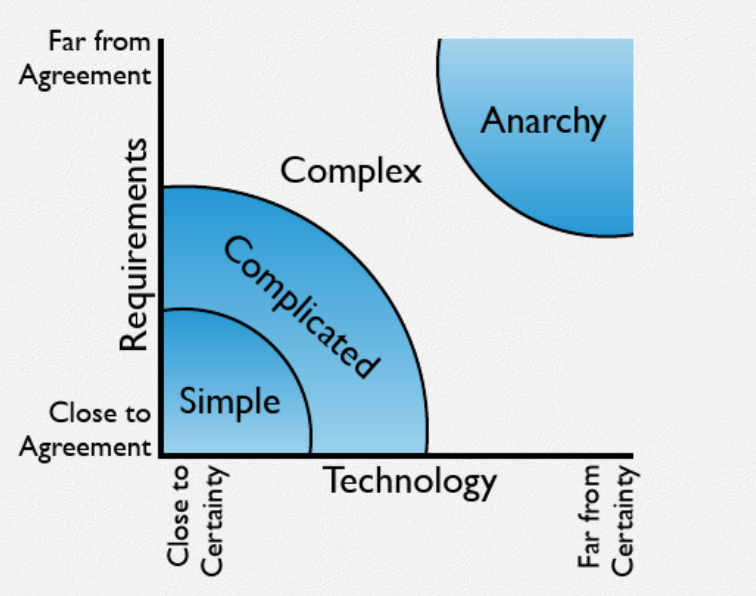


Figure 3.1: Project noise level

The scrum process is also scalable, since it to organize processes that require many people several team can be created and other scrum teams can be used to organize them, creating scrums of scrums.

Sprints Scrums processes make progress in a series of sprints (Fig. 3.2), which are similar to the iteration in other processes, however they have typical duration which is 2-4 weeks; in this time frame a product must be designed, implemented and tested, this time constraints is thought to give rhythm to the team.

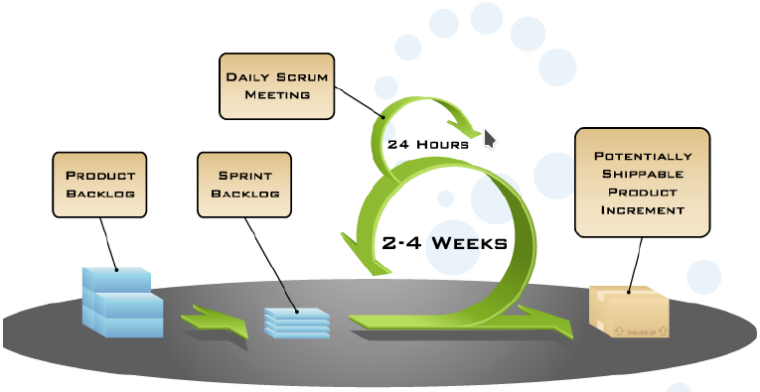


Figure 3.2: The workflow and the artifacts used and produced in a sprint

Sprints make scrum teams do a little of every phase for each sprint and in general during a sprint requirements cano be changed, so the duration of the sprint must take into account how long the team can avoid changes while

implementing a part of the project.

3.1 Scrum Framework

A scrum process has a certain framework which is composed of a set of role, some activities to be performed (called ceremonies) and documentation to be produced (called artifacts).

3.1.1 Roles

Product Owner. Represent the management interest in the success of the project, it defines the features, content and release date of the product. It is responsible for the project return of investment, can accept or reject work results and prioritizes the features according to their market value, at each sprint.

Scrum master. Represents management to the project and is responsible for enacting Scrum values and practices into the team. It also removes impediments and ensures that the team is fully functional and productive, enabling close cooperation across all roles and functions. Finally it shields the team from external interferences, making it focus on the sprint.

Team. The team members cannot change when a sprint is in progress and are general 5 to 9 people[1], they all should be full-time employed to guarantee their commitment to the project and should have different capabilities (programmers, designers ,testers etc.). Teams are self-organizing.

3.1.2 Ceremonies

Sprint planning. The sprint planning is composed of two events:

Sprint prioritization where the product backlog is evaluate and the sprint goal is selected

Sprint planning where the design to reach the sprint goal is selected and the sprint backlog is created and estimated from the items chosen from the product backlog.

In the sprint backlog every task has an estimated duration, and all the tasks have an high level design. The whole planning is done collaboratively by all the team.

Daily scrum meeting. It's a daily meeting of about 15 minutes[2] where there is a review by all the team of what has been done until now, what are the problems and what else needs to be done. The meeting is intended as a commitment among peers.

Sprint review. An informal meeting where the team presents the sprint results, usually it involves only a demo, with no slides.

[1] The right number to share a pizza
[2] In front of the coffee vending machine

Sprint retrospective. After the sprint review there is this last meeting where the team focuses on what where the problems in the last sprint, how to address them and what instead should be repeated or introduced since it has a positive effect on the sprint. This meeting usually doesn't last more than 30 minutes.

3.1.3 Artifacts

Product backlog It is the list of the desired work on a project (Fig. 3.3), organized in items, which are prioritized at the beginning of each sprint by the product owner according to the value they have to the users of the product.

Product Backlog
Estimating System Upgrade

Sprint	ID		Backlog Item	Owner	Estimate (days)	Remaining (days)
1	1	Minor	Remove user kludge in .dpr file	BC	1	1
1	2	Minor	Remove cMap/cMenu/cMenuSize from disciplines.pas	BC	1	1
1	3	Minor	Create "Legacy" discipline node with old civils and E&I content	BC	1	1
1	4	Major	Augment eachtbl operation to support network operation	BC	10	10
1	5	Major	Extend Engineering Design estimate items to include summaries	BC	2	2
1	6	Super	Supervision/Guidance	CAM	4	4
	7	Minor	Remove Custodian property from AppConfig class in globals.pas	BC	1	
	8	Minor	Remove LOC_ constants in globals.pas and main.pas	BC	1	
	9	Minor	New E&I section doesn't have lblCaption set	BC	1	
	10	Minor	Delay in main.releaseform doesn't appear to be required	BC	1	
	11	Minor	Undo modifications to Other Major Equipment in formExcel.pas	BC	1	
	12	Minor	AJACS form to be centred on the screen	BC	1	
	13	Major	Extend DUnit tests to all 40 disciplines	BC	6	

Figure 3.3: A real product backlog

User stories User stories are the Agile equivalent of use cases, they are 3x5 cards[3] where a goal is represented, with the following format:

As a [role]

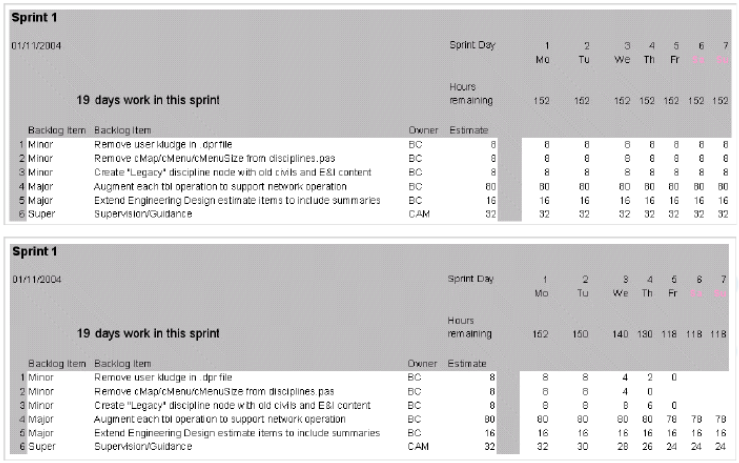
I want to [goal]

So that I can [reason/purpose]

Each of these cards has a value which is agreed on by the team, representing the perceived implementation difficulty of the story. The reason behind these format is that in this way the appear less complicated[4] to the customers, which can be involved directly in the product functionality description; they are also easy to rearrange during development.

Sprint backlog Team members sign up for work of their own choosing, the estimated work remaining is updated daily and Any team member can modify the sprint backlog (Fig. 3.4). Work for the sprint emerges as the sprint If work is unclear, define a sprint backlog item with a larger amount of time and break it down later, so work is updated as it becomes known.

[3] it's the first used format, a post-it, used by Connextra
[4] and scary, bhohohohoho



Sprint 1

01/11/2004

19 days work in this sprint

	Backlog Item	Backlog Item	Owner	Estimate	Sprint Day	1 Mo	2 Tu	3 We	4 Th	5 Fr	6	7
					Hours remaining	152	152	152	152	152	152	152
1	Minor	Remove user kludge in .dpr file	BC	8		8	8	8	8	8	8	8
2	Minor	Remove cMap/cMenu/cMenuSize from disciplines.pas	BC	8		8	8	8	8	8	8	8
3	Minor	Create "Legacy" discipline node with old civils and E&I content	BC	8		8	8	8	8	8	8	8
4	Major	Augment each tbl operation to support network operation	BC	80		80	80	80	80	80	80	80
5	Major	Extend Engineering Design estimate items to include summaries	BC	16		16	16	16	16	16	16	16
6	Super	Supervision/Guidance	CAM	32		32	32	32	32	32	32	32

Sprint 1

01/11/2004

19 days work in this sprint

	Backlog Item	Backlog Item	Owner	Estimate	Sprint Day	1 Mo	2 Tu	3 We	4 Th	5 Fr	6	7
					Hours remaining	152	150	140	130	118	118	118
1	Minor	Remove user kludge in .dpr file	BC	8		8	8	4	2	0		
2	Minor	Remove cMap/cMenu/cMenuSize from disciplines.pas	BC	8		6	6	4	0			
3	Minor	Create "Legacy" discipline node with old civils and E&I content	BC	8		8	8	8	6	0		
4	Major	Augment each tbl operation to support network operation	BC	80		80	80	80	80	78	78	78
5	Major	Extend Engineering Design estimate items to include summaries	BC	16		16	16	16	16	16	16	16
6	Super	Supervision/Guidance	CAM	32		32	30	28	26	24	24	24

Figure 3.4: A sprint backlog example

Burndown chart A display of what work has been completed and what is left to complete, for each developer or work item. It is update every day with the remaining amount of estimated hours needed to complete the sprint.

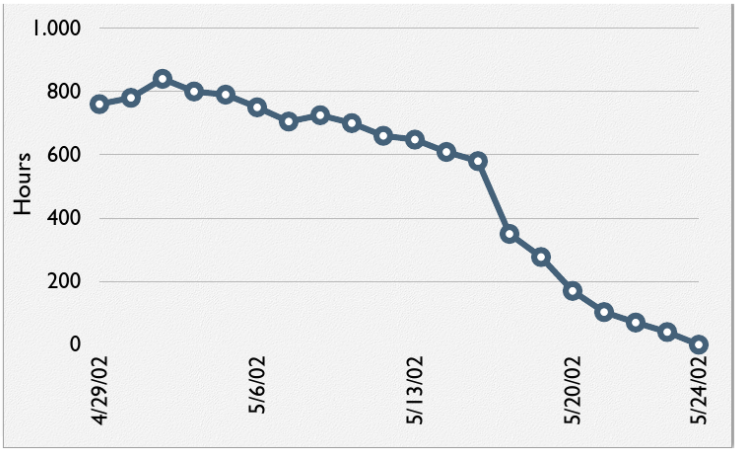


Figure 3.5: Burndown char example

Release burndown chart A variation of the burndown chart where the overall progress is shown, it is updated at end of each sprint.

Chapter 4

Distributed Programming

4.1 Client-Server Communication

Layered architecture The layered architecture (Fig. 4.1) separates layers of components from each other, giving it a much more modular approach. A well known example for this is the OSI model that incorporates a layered architecture when interacting with each of the components. Each interaction is sequential where a layer will contact the adjacent layer and this process continues, until the request is been catered to.

Definition. *A client is any user or program that wants to perform and operation on the system.*

1. Clients interact with the system through a presentation layer.
2. The application logic determines what the system actually does.
3. The resource manager deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic.

To indicate these layer there are several terminologies, in many cases the client concept includes also the presentation layer.

Multitier architecture Multitiered architecture is a client–server architecture in which presentation, application processing, and data management functions are physically separated. Tiers are different from layers since they refer to the physical organization on the system.

Common multitier architectures are:

One-tier: Everything happens in the same machine, easy to optimize and there are no forced control switches. (Fig. 4.2)

Two-tiers: Clients have their own presentation layer, so we can differentiate presentation layers between clients and save system resources, allowing the resource manager to allocate resources only for the application layers, it also allow systems federation. (Fig. 4.3)

Three-tiers: The application logic, presentation layer and resource manager are each on different machines. (Fig. 4.4)

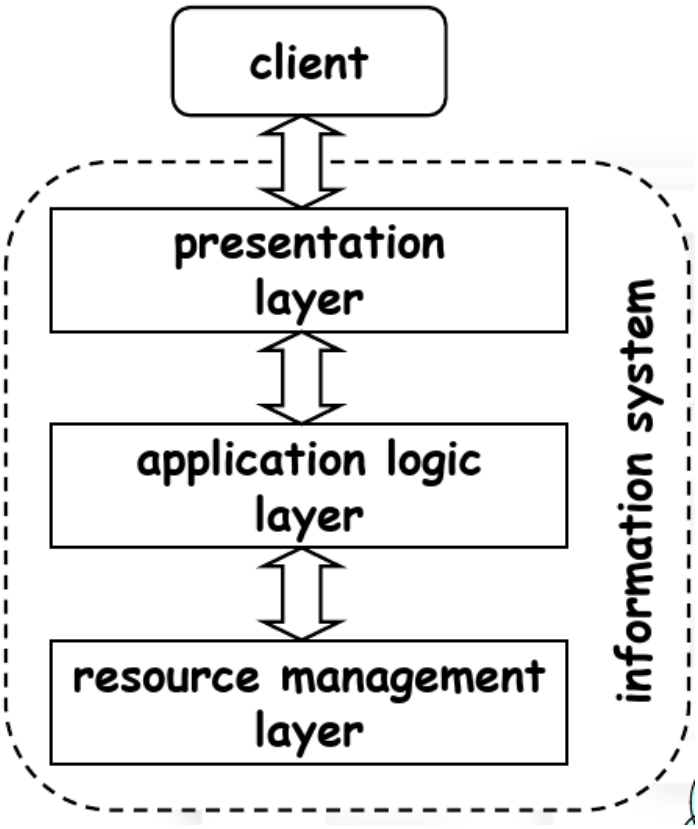


Figure 4.1: A layered distributed system

N-tiers: Generalization of three-tier architecture which has more tiers.

With three or more tiers we need a piece of software that will act as an interpreter between the application logic and the resource manager and the client, since the application layer will receive requests from both these subsystems or more in a N-tier architecture.

Definition. *Application Program Interface: An interface to invoke the system or a service of the system from the outside.*

4.2 Middleware

Definition. *Middleware is a level of indirection between clients and other layers of the system, which introduces an additional layer of business logic encompassing all underlying systems.*

It has two different communication primitives:

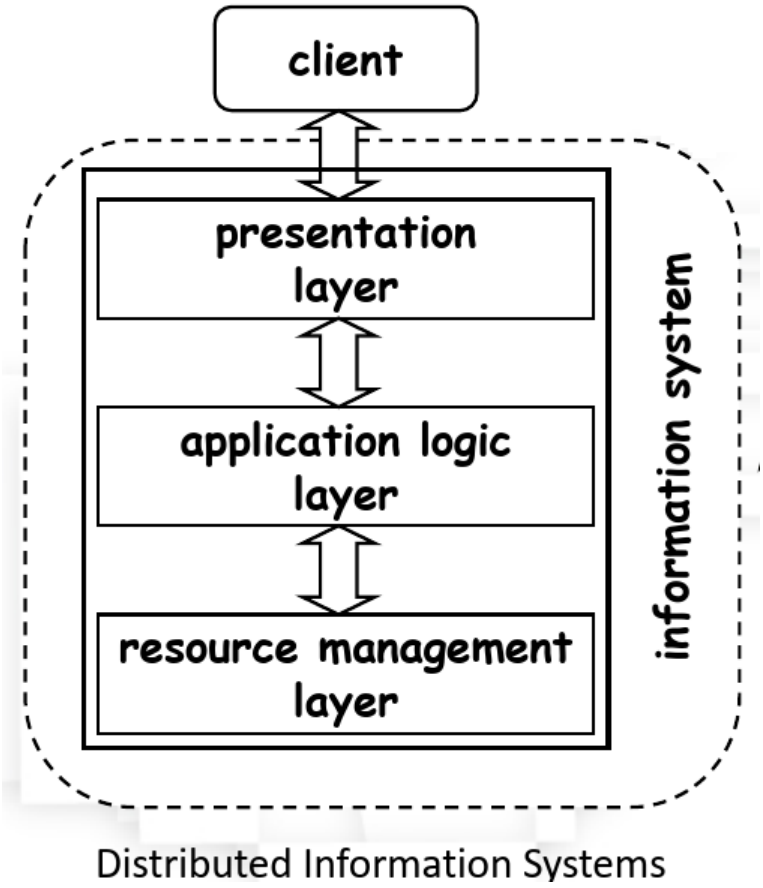


Figure 4.2: One tier architecture

Blocking interaction Which is an improperly synchronous interaction where the client is blocked until the response is received (without having a bounded time for execution hence improperly synchronous).

Non-blocking interaction The client can continue its execution without explicitly waiting for the response.

Blocking interaction. Traditionally, distributed applications use blocking calls. Synchronous interaction requires both parties to be "alive", the caller must wait until the response comes back. The receiver does not need to exist at the time of the call (e.g., some technologies create an instance of the service/server /object when called if it does not exist already).

Because it synchronizes client and server, this mode of operation has several disadvantages:

- connection overhead
- higher probability of failures

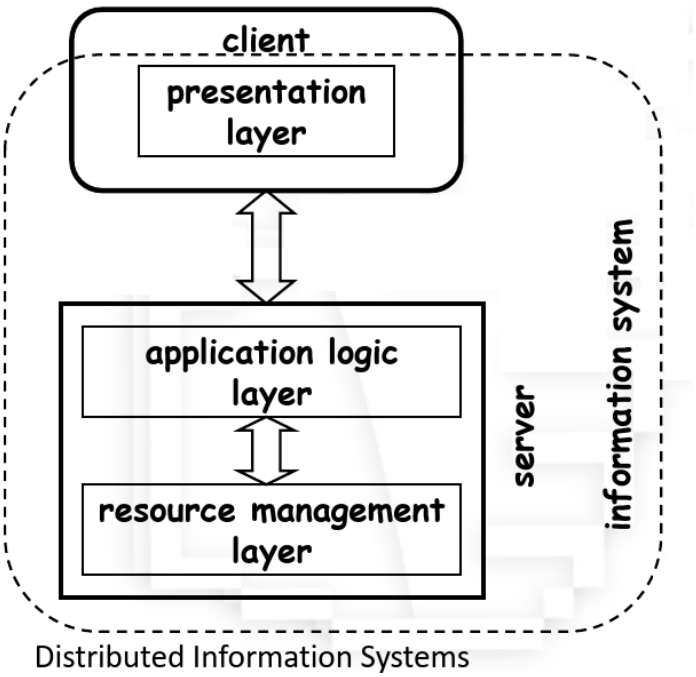


Figure 4.3: Two tier architecture

- difficult to identify and react to failures
- it is a one-to-one system; it is not really practical for nested calls and complex interactions, which increase these problems

Synchronous invocations require to maintain a session between the caller and the receiver, which is expensive in terms of CPU resources, limiting the number of active sessions. An improvement is session pooling, where a pool of session is maintained active and for each request only a thread is associated with a session. There is also the need of a context management system which will associate each client with the proper responses and the relative data to make meaningful interactions.

A crash can create several problematic situations, because the context is lost and needs to be reinizialized:

- If the client crashes before sending the request we have no problems.
- If the server crashed we can have three possibilities:
  - the crash is before the request processing, we lose the request
  - the crash is during the request processing, we can have inconsistencies
  - the crash is before the response is sent, the client will not see any response but the request has been performed.

To solve these problems the middleware provides two solutions:

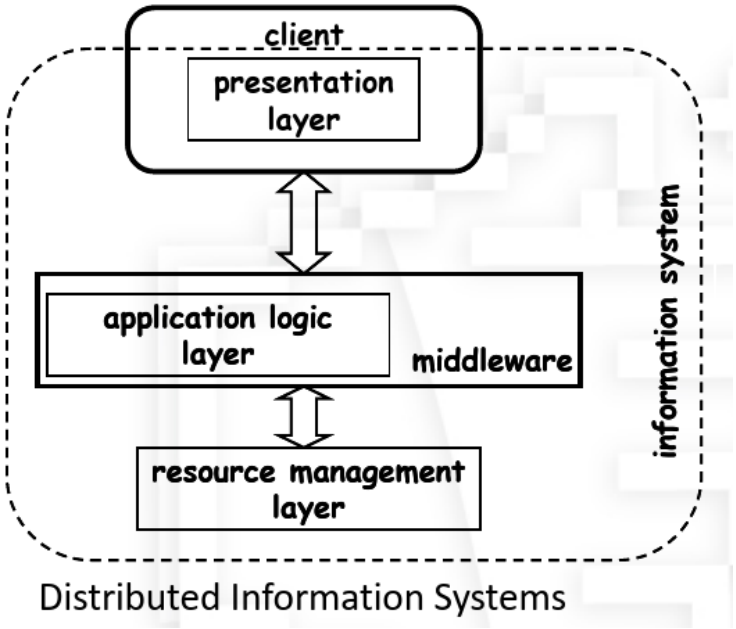


Figure 4.4: Three tier architecture

1 - Enhanced support.

Transactional interactions: which enforce the exactly one semantic for the procedure calls giving execution guarantees that enable more complex interactions.

Service replication and load balancing: preventing the service from becoming unavailable in case of failure (does not address the failure recovery)

2 - Asynchronous interaction. That can take place in two forms and keep the request stored until a response is received:

non-blocking invocation: where the client does not explicitly wait for the response

persistent queues: request and response are stored persistently in a queue until they are both accessed by client and server.

Definition. *Remote Procedure Call: hides communication details behind a procedure call and helps bridge heterogeneous platforms*

Definition. *Sockets: operating system level interface to the underlying communication protocols*

Definition. *TCP, UDP: User Datagram Protocol (UDP) transports data packets without guarantees. Transmission Control Protocol (TCP) verifies correct delivery of data streams*

Definition. *Internet Protocol (IP): moves a packet of data from one node to another*

Understanding Middleware Middleware can be intended as:

Programming Abstraction: to hide low level details of hardware, networks, and distribution; the trend is towards increasingly more powerful primitives that, without changing the basic concept of RPC, have additional properties or allow more flexibility in the use of the concept. The programming model of a middleware infrastructure shows the limitations, performance and evolution of the platform, giving insight on its applicability.

Infrastructure: to provide a comprehensive platform for developing and running complex distributed systems, which result in service oriented architectures with standardized interfaces. Which takes into account the non functional properties ignored by data models (security, performance, etc.) and make development, maintenance and monitoring cheaper.

4.2.1 Message oriented middleware

Message oriented middleware (MOM) uses a two queues to communicate act as an interface between the parties which need to communicate, these queues are shared and the whole infrastructure communicates with messages. This type of interaction creates a new type of interaction, an anonymous many to many interaction.

Publish/Subscribe interaction. Partecipans are divided in two groups

- Publishers (which send messages)
- Subscribers (which receive messages)

The middleware acts as the central entity which receives the messages from publishers and relays them to the interested subscribers, the communication can be topic bases (subscribers notify their interest in a set of topics) or content based (more accurate filters are used to check the content of the message). Notification can be given by the middleware (push model) or the subscribers can explicitly check if there are new messages (pull model).

4.2.2 Middleware on the internet

In some cases infrastructures use a public network to connect some entities and in this case the firewall can be a problem since they are configured to block RPCs. To make the RPC work with firewall, thus allowing only trusted call we have several options, however the best option is to tunnel these requests, which can involve a server, a client and the middleware (which is split between server and client). (Fig. 4.5)

4.3 Remote Procedure Calls

RPC is a point-to-point protocol which allows two entities to communicate when there are more than two entities communicating; each call is treated as independent (but they could be not independent), plain RPC make recovery from a failure difficult.

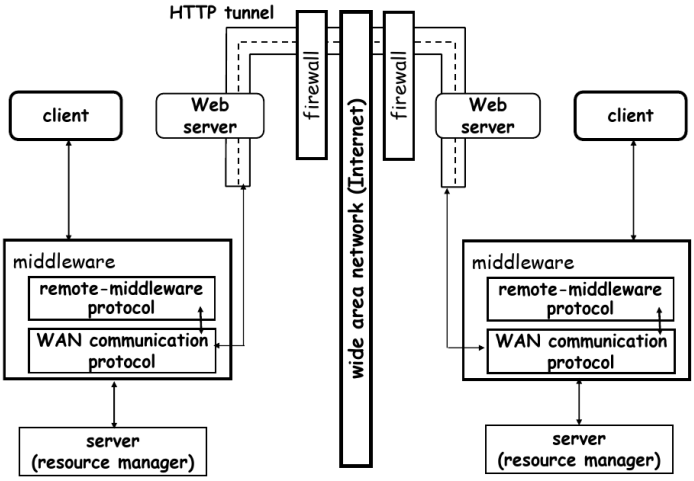


Figure 4.5: Infrastructure over a public WAN tunnelling middleware communication

To use an RPC we need client and server stubs to be create by the development environment, using the interface definition language. This makes both client and server have the impression that the procedure is called on the local machine. (Fig. 4.6)

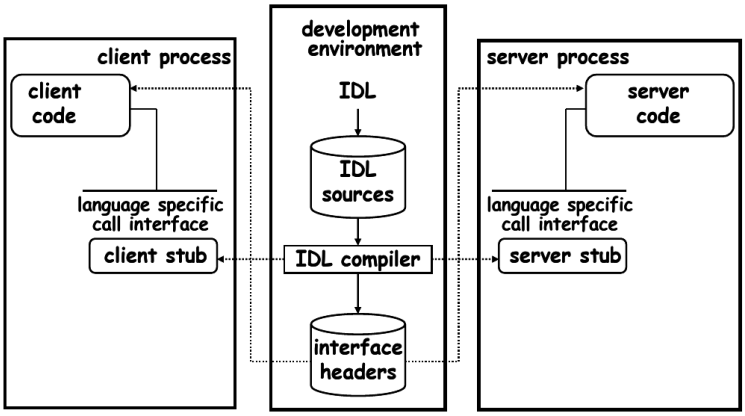


Figure 4.6: Stubs creation

Then the communication can be done directly between client and server (Fig. 4.7)

or by relying on the middleware which can also act as a name server to client, balancing the load between the servers and offering some execution guarantees. (Fig. 4.8)

With RPC we can avoid to implement a complete infrastructure for every distributed application and:

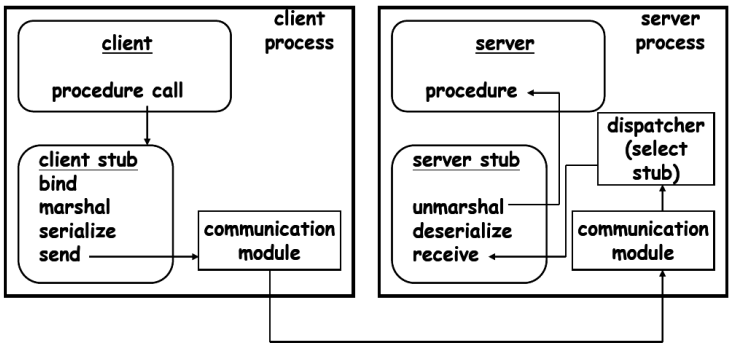


Figure 4.7: Direct communication

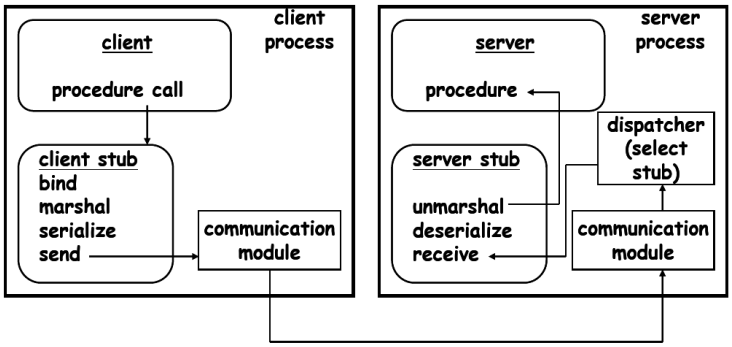


Figure 4.8: Middleware-based communication

- Hide distribution behind server procedure process calls
- Provide an interface definition language (IDL) to describe the services
- Generate all the additional code necessary to make a procedure call remote and to deal with all the communication aspects language
- Provide a specific binder in case it has a distributed interface name and directory service system

4.3.1 Transactional RPC

Transactional RPC provide the language to bind several call into a single atomic unit (Fig. 4.9), usually includes an interface to databases to execute transactions using the 2 phase commit.

Two phase commit. Uses a transaction monitor which before committing the transaction check if all interested parties can execute the update and after

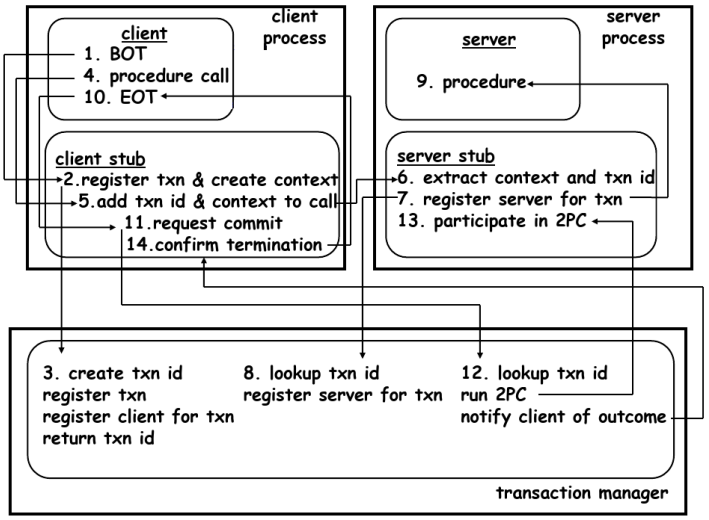


Figure 4.9: Transactional RPC communication

receiving all the responses decides to commit or rollback the transaction.

Chapter 5

Introducing Web Services

Definition. *A web service is a software component available on the Web, a programmatically available application logic exposed over the Internet. Any piece of code and any application component deployed on a system can be transformed into a network-available service. A Web service can be invoked by some other client application/component.*

Definition. *e-Service is the provision of a service via the Internet, meaning just about anything done online. Basically whichever Web application usable by a human, through a user interface. To build an e-service several web services may be needed.*

Services reflect a new "service-oriented" approach to programming, based on the idea of composing applications by discovering and invoking network-available services rather than building new applications or by invoking available applications to accomplish some task.

5.1 Web Services

A web service performs some encapsulated business function which can be:

- a self-contained business task (e.g. funds withdrawal);
- a full-fledged business process (e.g. automated purchasing of supplies);
- an application (e.g. demand forecasts);
- a service-enabled resource (e.g. access to back-end database);

Web services can be mixed together to create a complete process (Fig. 5.1) with decreased human interaction, this process is platform independent and can have several billing models based on the service performance and/or subscription.

Application Service Provider. The concept of software as a service appeared for the first time in the ASP model, where an application is rented to a client. This application is developed by in all its aspects to provide a complete solution in which the client has little opportunity to customize it (only appearance can be changed) and it can be hosted by the developer or can be integrated as a module in the customer's website.

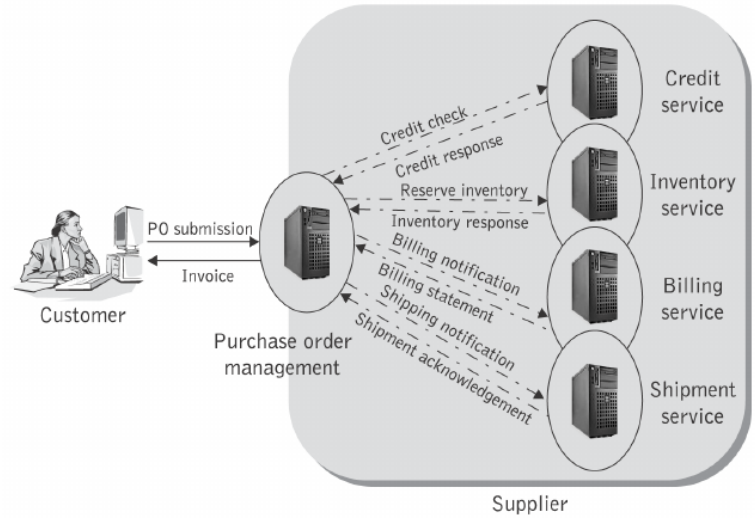


Figure 5.1: A web service based process

ASP vs web services. The ASP model has several limitations, such as:

- inability to build a highly interactive application;
- inability to build a complete customizable application;
- inability to integrate several applications;
- the ASP model uses monolithic application, in which modules are hard to be decoupled and reused;

On the other hand the web service model as several characteristics:

- XML based;
- loosely coupled asynchronous interactions
- uses standards to make access and communication over the internet easier

Web service can be used between an enterprise to save development costs and improve reusage of the same components or between enterprises to have a standard mean to provide and access services, improving the business opportunities.

5.1.1 Web services properties

Web services can be:

- Simple informational services which expose an API or have a request-response behavior;
- complex services which depend on pre-existent services to create multi-step business interaction;

The service is characterized by functional or non-functional properties and can be stateless or stateful; they can have a fine (almost an atomic behavior) or coarse granularity (a complex and possibly multi-session behavior). All web services are loose coupled, since they do not need to know how their partner in communication behaves or is implemented. The interaction with a web service can either be synchronous (RPC-like with simple information, e.g. only values) or asynchronous (message-style service where documents are exchanged). In either case the interaction is well defined by the web service description language (WSDL) which describes the rules use to interfacing and interacting with between the web service and other applications. The implementation (Fig. 5.2) of the web service is concealed by a service interface which is described by the WSDL and can be implemented in different languages according to the service provider, however it always describes itself and behaves in the same way, so it can be accessed by the whole world.

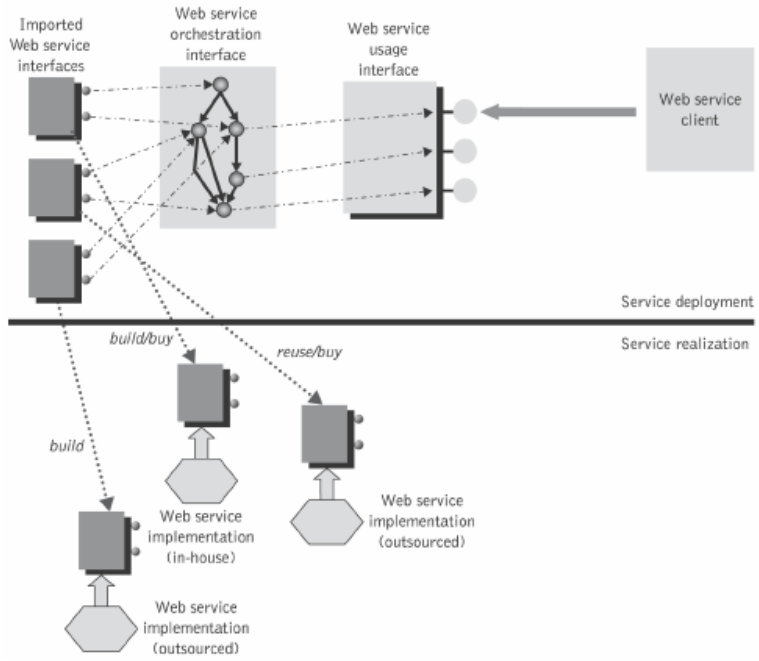


Figure 5.2: Web services realized by different means (bottom) and deployed to be used by a client (top)

5.1.2 Roles

The service model uses three different roles (Fig. 5.3):

Service provider: organizations that provide the service implementations, supply their service descriptions, and provide related technical and business support;

Service clients: end-user organizations that use some service;

Service registry: a searchable directory where service descriptions can be published and searched by service requestors to find service descriptions and obtain binding information for services. This information is sufficient for the service requestor to contact, or bind to, the service provider and thus make use of the services it provides.

In this way we can design a software system that provides services to clients (which can be other applications or end users) that are distributed in a network via a published discoverable interface. These web services use several related

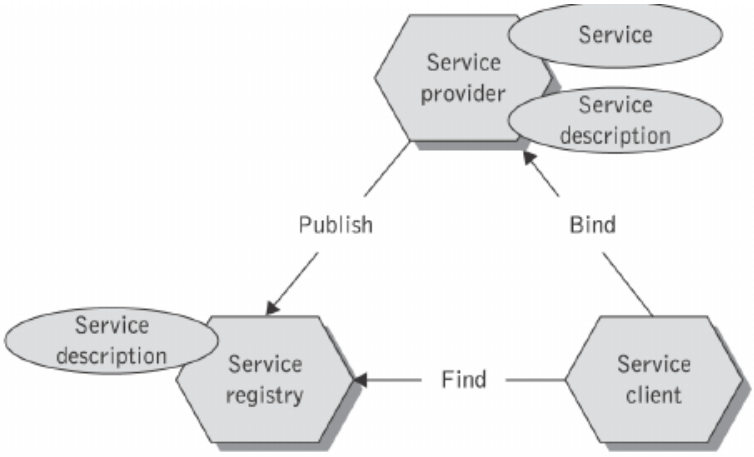


Figure 5.3: Interaction flow of the roles used in web services applications

technologies and standards that are used a stack to provide the necessary functions.

Definition. *QoS refers to the ability of a Web service to respond to expected invocations and perform them at the level commensurate to the mutual expectations of both its provider and its customers.*

Definition. *An SLA is a formal agreement (contract) between a provider and client, formalizing the details of use of a Web service (contents, price, delivery process, acceptance and quality criteria, penalties, etc in measurable terms) in a way that meets the mutual understandings and expectations of both providers and clients.*

Web services offer several advantages:

- Standard way to expose legacy applications;
- Overcome application integration issues;
- standard way to develop internet native applications for every internal and external usage;
- standard interface for cross-enterprise specific systems, to ease business to business integration;

5.2 Communication with SOAP

Conventional distributed application use distributed communication technologies based on Object oriented RPC to couple network protocols and object orientation (ORPC use a symbolic name to locate an object in the server process). These technologies however require that the both ends of the communication are implemented in the same language and can be blocked by firewalls since they have proprietary protocols that may not be considered safe in some networks.

SOAP is an XML-based communication protocol which allow different entities (implemented in different technologies) to communicate over the network. It uses XML to code a request-response scheme that is carried over the network by HTTP and its goal is inter application communication. (Fig. 5.4)

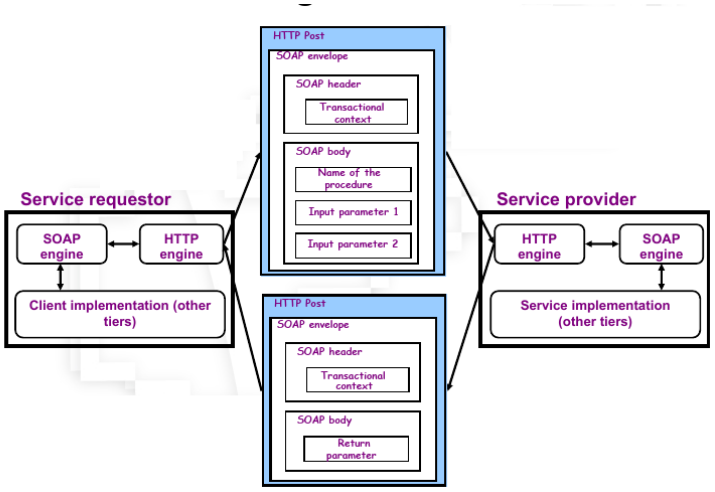


Figure 5.4: Soap based communication (RPC request/response with POST method)

SOAP can be used in two ways:

RPC communication: Uses XML to code bot the request and the response;

Document exchanging: The document is coded with XML and there may not be a response to a message carrying this document, the protocol leaves the contents of the document untouched;

The default binding for the SOAP protocol is the HTTP, however other protocols can be used to carry SOAP messages, but HTTP is preferred since they use the same error codes and HTTP is permitted in almost every network. SOAP with HTTP can use both GET and POST operations, with GET only the response is a SOAP message but when POST is uses the request and the response are both SOAP messages.

Advantages of SOAP are:

- Simplicity
- Portability

- Firewall friendliness
- Use of open standards
- Interoperability
- Universal acceptance.

Disadvantages of SOAP are:

- Too much reliance on HTTP
- Statelessness
- Serialization by value and not by reference.

5.3 Service description and WSDL

Web services must be defined in a consistent manner to be discovered and used by other services and applications. Web service consumers must determine the precise XML interface of a web service because the XML Schema alone cannot describe important additional details involved in communicating with a Web service. The service description reduces the amount of required common understanding and custom programming and integration being a machine understandable standard describing the operations of a Web service that specifies the wire format and transport protocol that the Web service uses to expose this functionality and can also describe the payload data using a type system.

Definition. *The web services description language (WSDL) is the XML-based service representation language used to describe the details of the complete interfaces exposed by Web services and thus is the means to accessing a Web service.*

In detail the WSDL uses an XML document that describes how to interact (Fig. 5.5) with its associated web service and it is a "contract" between the provider and the requestor, since it binds both of them to the described interaction.

WSDL is an platform independent language that describes:

What a service does;

Where it resides;

How to invoke it;

WSDL documents can be separated into distinct sections (Fig. 5.6):

- The service-interface definition that describes the general Web service interface structure. Comprehends: data type, operation parameters, set of operations and action descriptions. This contains all the operations supported by the service, the operation parameters, and abstract data types.

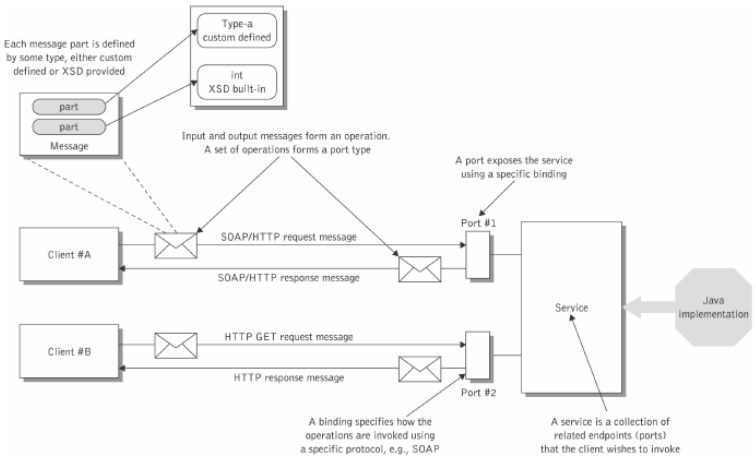


Figure 5.5: WSDL influence over communication

- The service implementation part binds the abstract interface to a concrete network address, to a specific protocol, and to concrete data structures. It describes bindings operation, endpoints associated with the binding and location for each binding, it can also reference other XML documents.

In this way each part can be independently reused and their combination provide sufficient information for the service requestor to use the service.

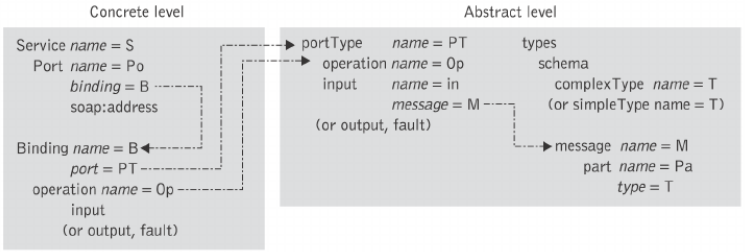


Figure 5.6: WSDL document parts and their connections

WSDL interfaces support four common types of operations that correspond to the incoming and outgoing versions of two basic operation types:

- An incoming single message passing operation and its outgoing counterpart ("one-way" and "notification" operations)
- the incoming and outgoing versions of a synchronous two-way message exchange ("request/response" and "solicit response").

Any combination of incoming and outgoing operations can be included in a single WSDL interface, providing support for both push and pull interaction models at the interface level.

5.4 Registry and UDDI

Universal description, discovery and integration is a registry standard which maintains the web services description and provides a registry that supports service publishing and subscription. Enabling (Fig 5.7):

- service and business description;
- discovering of business that offer a service;
- integration with the desired businesses (through service usage)

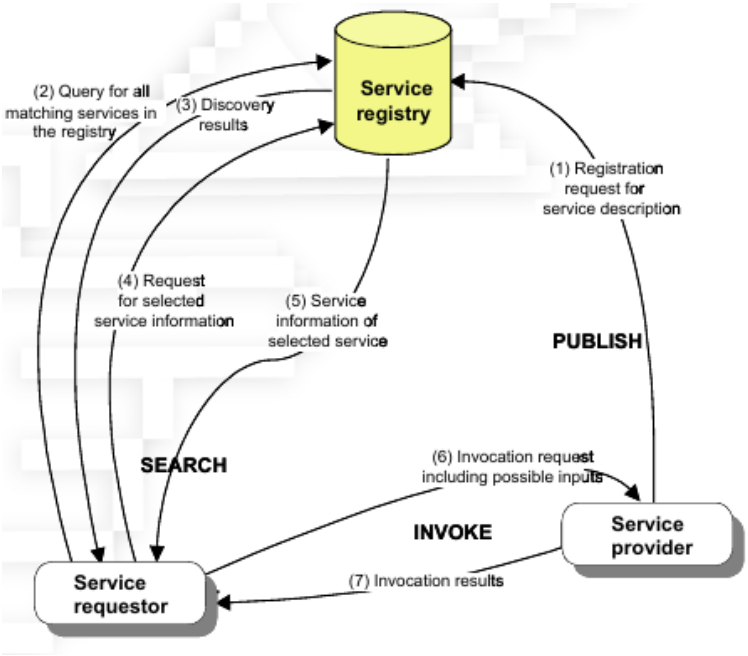


Figure 5.7: UDDI registry usage

UDDI business registration consists of three components:

white pages: contact information;

yellow pages: classification information
(based on standard industry taxonomies);

green pages: services information and technical capabilities;

Web services have many advantages, however they are not the solution to everything since they have several disadvantages:

- performance issues
- no appropriate transaction management
- cannot express business semantics
- too many unharmonized emerging standards that can compromise mission critical operations

5.5 RESTful services

REST is a simple protocol that transmit data over HTTP without the SOAP additional layers (Fig. 5.8), the data is in the form of web page meant to be consumed by a program.

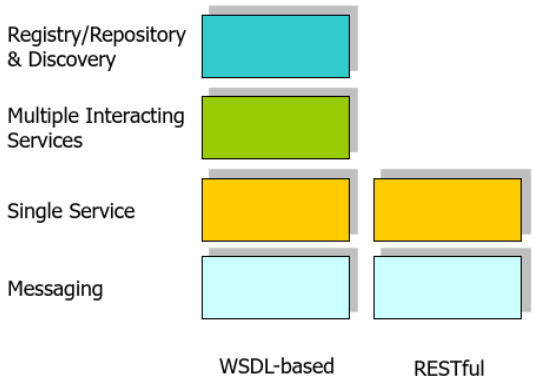


Figure 5.8: WSDL and RESTful architectures comparison

RESTful actions are formed by an URIs and an HTTP verb (Fig. 5.9).

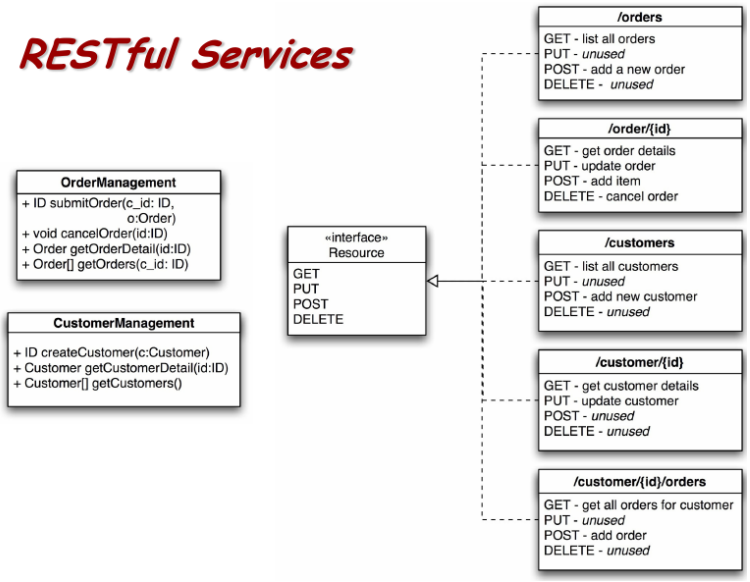


Figure 5.9: RESTful command mapping example

RESTful interactions follow some simple basic principles:

- Addressability (and URL for each resource)
- Uniform Interface

- Stateless interactions
- Self-describing messages
- Hypermedia

REST and RPC are incompatible since REST addresses end-points and RCP address software components. Compared to SOAP, REST is way lighter but both of them provide the same functionalities.

Chapter 6

Microservices

Microservices [1, 2, 3, 4] are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.

6.1 Monolithic vs. Microservices Architecture

Monolithic Architecture. During the client/server era, we tended to focus on building tiered applications by using specific technologies in each tier. The term monolithic application has emerged to describe these approaches. The interfaces tended to be between the tiers, and a more tightly coupled design was used between components within each tier. Monolithic applications are often simpler to design, and calls between components are faster because these calls are often over interprocess communication (IPC). Also, everyone tests a single product, which tends to be a more efficient use of human resources. The downside is that there's a tight coupling between tiered layers, and you can't scale individual components. If you need to do fixes or upgrades, you have to wait for others to finish their testing. It's harder to be agile.

So with a monolithic architecture we have a more efficient systems that is easy to manage with these disadvantages:

- The architecture is hard to evolve
- Long time for each release (Bigger the system means longer build, test and release cycle)
- Long time to add new feature (a new feature must be made compatible with all the existing ones)

These disadvantages origin from the tight coupling that the module have in the monolithic architecture and from the use of shared libraries between many different services, this means that an update on these libraries requires the adaptation of all the service that use these libraries. The same happens with the data, because they are likely to be stored in a single database in several tables, permitting developers to create dependencies between tables and to stress the DBMS. An application of this type is scaled only by cloning it on multiple servers-

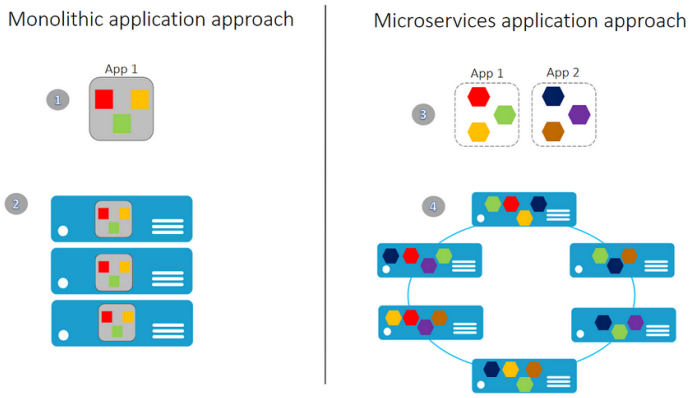


Figure 6.1: Microservices and Monolithic architecture comparison

Microservices architecture.

Definition. *Microservices is a service-oriented architecture composed of loosely coupled elements, each with a bounded context.*

Microservices are self-contained piece of business functionality with clear interfaces, and may, through its own internal components, implement a layered architecture. Each one typically encapsulates simpler business functionality, which you can scale up or down, test, deploy, and manage independently.

Each microservice is independent from the others and it is developed by a dedicated team, so the updates of a microservice are independent from the other microservices since the only requirement for the communication is an interface exposed over the network. (Fig. 6.1)

One important benefit of a microservices approach is that teams are driven more by business scenarios than by technology. Smaller teams develop a microservice based on a customer scenario and use any technologies that they want to use.

This means that there is no need for standardized tech to be used in the whole organization, however a set of preferred technologies is recommended.

Each microservice has a small data storage (a database with the chosen technology for the service), an application level and a public API exposed over the network, to avoid coupling with other services.

When you use a microservices approach, you compose your application of many small services. These services run in containers that are deployed across a cluster of machines. There is also an orchestrator that has the job of coordinating all these services to execute correctly your application logic.

6.2 Microservices Characteristics

Microservices follow some basic principles:

Reliance only on the public API. (Rely on the public API) This allows to simplify the communication and to decouple the communication from the application logic and the format in which data is stored, however the API must

evolve in a backward compatible way, to give the other services the time to adapt to the API changes.

Using of the best tool for the each microservice (Use the right tool for the job) In a microservice we can (and should) use the best technology we can afford to accomplish the scope of the service, since other services can see only the API, moreover we can also change the used technologies as we like without worries, since we must only maintain the API functionalities.

Services must be secured. (Secure your services) Communication over the network should not be unprotected, even if it is between microservices (since the can be scattered around the word), so there are several means to protect the data and the availability of microservices like API throttling, sandboxes, gateways, authorization , authentication and sessions/secrets management.

Microservices should create a good ecosystem.(Be a good citizen within the ecosystem) Developing several independent microservices is done to increase the efficiency of an application or service and we must keep in mind that these services are not visible to the end user, so we must define the requirements for each service in a way that they are compatible and compliant with the application SLA and the user expectancies. To do so we need to log each microservice to understand if it is meeting the requirements and we should also create an application log to see what and where there can be problems when all the microservices are used as an ecosystem.

More than a technology transformation. Shifting from the monolithic to the microservices architecture will likely have and impact in the organization's structure, since now there are small teams that need the same things that were needed by the monolithic application, so there cannot be anymore functional teams (e.g. a team composed only of UI specialists), since the need for a specific functionality is now distributed across all the microservices. This also means that teams are smaller and there is better accountability and focus on the scopes of each functionality since each team owns a microservice.

Automation. (Automate everything) The best way to increase the productivity with microservices is to use an infrastructure, like Amazon AWS or Microsoft Azure, because these infrastructure provide several functionalities to automate the deployment processes, the scaling and the maintenance of each microservice.

So the use of microservices has several benefits, even if it is not applicable everywhere:

- Easier the to maintain and evolve (we can evolve each component independently)
- Faster Build/Test/Release cycle
- Easy to add new functionalities (the accountability and ownership is clear)

Chapter 7

Measurements and statistics

Measures, in the field of Software Engineering, are meant to monitor a software project in various aspects: (Fig.7.1)

- Verifying how far quality parameters are from reference values
- Identifying deviations from temporal and resource allocation planning
- Identifying productivity indicators
- Validating the effect of strategies aimed at improving the development process (quality, productivity, planning, cost control)

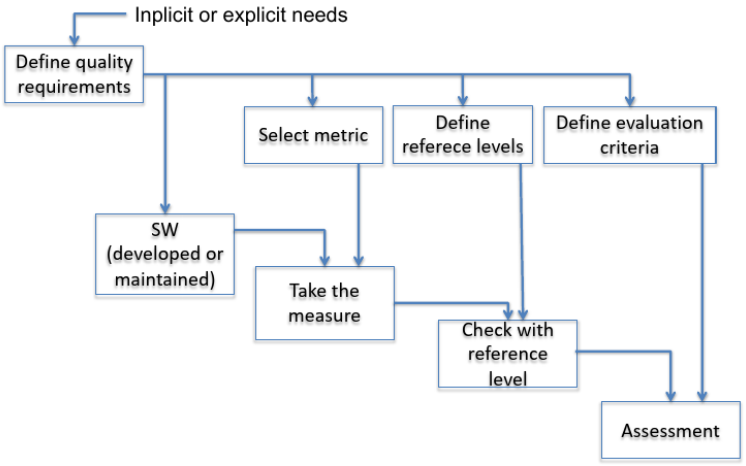


Figure 7.1: Monitoring a software project according to the selected criteria

After having took the needed measures we need to map them in a a reference scale, which will provide a qualitative rating of the project performance (Fig. 7.2), because measures themselves do not provide any quality judgment.

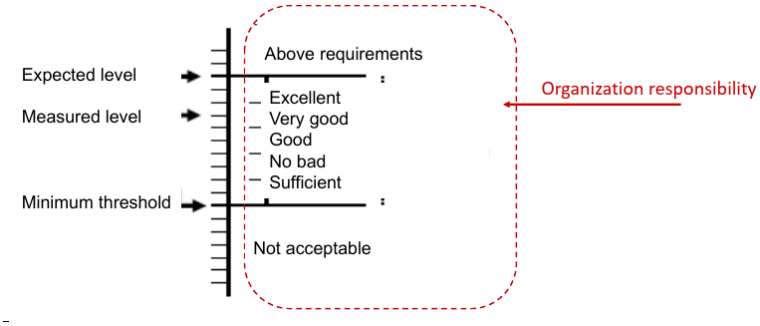


Figure 7.2: Mapping of measurements to a qualitative scale

7.1 Basics of measure theory

For our purpose we consider only five different measure scales (Fig. 7.3):

Nominal scale: Classifies object into two categories, where members of the same category share some characteristics and each element ca be only in one category (operators supported $\{=, != \}$). Its average values is meaningful only with frequency analysis on elements that appear in a certain category.

Ordinal scale: It allows us to classify and rank objects in the scale, but we still don't know the distance between any pair of ranks (operators supported $\{=, != , >, <\}$). In some ordinal scales the distance between items is assumed to be careful but not always an average has sense.

Interval scale: An interval scale allows us to classify and rank items, knowing the sizes of differences between them, they normally have an arbitrary minimum and maximum and are characterized by having equal intervals between ranks. A scale like this also need the definition of the used unit measure and cannot measure the magnitude between two values, since the zero point is not absolute (operators supported $\{=, != , >, <, +, -\}$).

Ratio scale: A interval scale with an absolute zero value, to support magnitude comparison (operators supported $\{=, != , >, <, +, -, *, /\}$).

Absolute scale: Ratio scale without values below zero.

The scale to be used in measurement must be chosen according to the characteristics of the objects we are measuring.

7.1.1 Types of measures

Definition. *Ratio: division between two values that come from two different and disjoint domains.*

Ratio is usually multiplied by 100 but is not a percentage, (e.g. males/females).

Definition. *Proportion: division between two values where the dividend contributes to the divisor.*

Scale types	Admissible transformations	Basic empirical operation	Appropriate statistical indexes	Appropriate statistical tests	EXAMPLES
NOMINAL	any one-to-one transformation	equality test	Mode Frequency	not parametric	labeling classify
ORDINAL	M(x)≥M(Y) implies that M'(x)≥M'(Y)	equality test and > <	Median Percentiles Spearman *r* Kendall W Kendall T	not parametric	preferences ordering di entità
INTERVALS	M'=aM+n(a>0) [*positive, linear*]	equality test and > + and -	Aritmetic mean Standard deviation Pearson correlation Multiple correlation	not parametric	Fahrenheit o Celsius date time
RATIO	M'=aM (a>0) [*similarity transformation*]	equality test and > + and - * and /	Geometric mean Armonic mean Coefficiente di variazione Percentage variation Correlation index	not parametrico and parametric	time intervals Kelvin lenghts
ABSOLUTE	M'=M [*identity*]				entity count

Figure 7.3: Comparison between the various scales

It's a value in $[0, 1]$ (e.g. a/(a+b)).

Definition. *Percentage: a proportion or fraction expressed normalizing the divisor to 100.*

It should be avoided if values are below 30-50%

Definition. *Rate: Measures the changes of quantity in respect to a quantity on which it depends.*

Definition. *Working definition: A debatable, non ambiguous and measurable definition.*

Working definition a used to validate statements with the usage of measures and a statistical analysis, since a statement can be not measurable (Fig. 7.4).

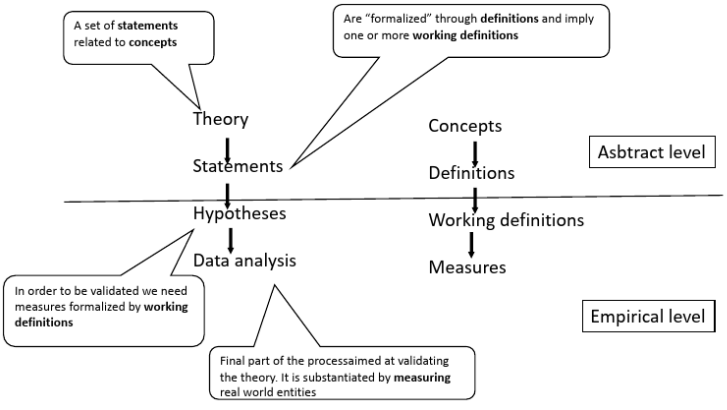


Figure 7.4: Different levels of abstraction and their relations

7.1.2 Quality of a measure

Reliability of a measure. The reliability of a measure is estimated with the value of σ^2 on repeated measures of the same value. It indicates if the measure can be repeated and the consistency of it.

Validity of a measure. The best available approximation to the truth or falsity of a given inference, proposition or conclusion. Combining validity and reliability of a measure we can tell how "good" was the measure we took.

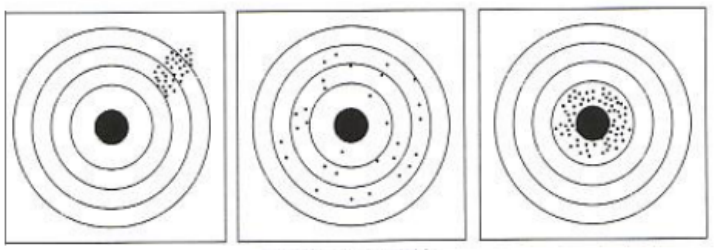


Figure 7.5: How validity and reliability of a measure are used to evaluate the "goodness" of the measure

Errors in measuring We know (by experience) that if we measure the same object twice we could get different results, we model this phenomenon by defining the measure in the following way:

Definition. *Measure: A measure is composed by the true value of the phenomenon and the total error experienced while taking the measurement.*

$$M = T + E_{tot}$$

The total error experienced is broken down in two components [5]:

Systematic error: Influences validity of the measure, happens every time we take the measure (e.g. instrument's error), taking several measures and computing the average reduces this type of errors.

Random error: Influences reliability of the measure, has random behavior and it's unpredictable.

The reliability of the error is measured by computing the ratio between the the variance of the computed measure and the variance of the metric used.

$$\rho_m = \frac{var(T)}{var(M)} = \frac{var(M) - var(E_r)}{var(M)} = 1 - \frac{var(E_r)}{var(M)}$$

Correlation. Correlation indicates if two variables have some kind of linear relationship, it's a value between $[-1, 1]$ and it is 1 is the variables have the same behavior, -1 the behavior of one variable is the opposite of the other and 0 if the variables have no linear relation. (Fig. 7.6)

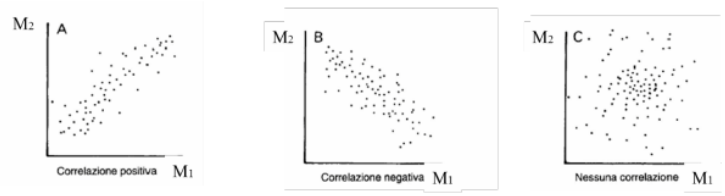


Figure 7.6: Examples of data with positive (A), negative (B) and no correlation (C)

7.2 Statistics

We analyze a population $M = x_1, .., x_n \subset X$ elements

7.2.1 Descriptive statistics

With descriptive statistic consider M to be our dataset, so we can identify several parameters:

Definition. *Mean:* $\mu = (\sum_i^n x_i)/n$

Definition. *Variance:* $\sigma^2 = [\sum_i^n (x_i - \mu)^2]/n$

Definition. *Standard deviation:* $\sigma = \sqrt{\sigma^2}$

Definition. *Median: The middle data point in a population; requires an ordinal scale to be computed and it's equivalent to the second quartile.*

When finding the median, if we have an even number of points we could have two medians; if we are in an interval scale we can take the average of the two.

Definition. *Percentile: value below which a certain percentage of the observations falls.*

Definition. *Quartile: one of the three values that divide the dataset into four equal parts.*

Definition. *Mode: the most recurring data point.*

Definition. *Range: distance between the smallest and largest data point.*

The frequency distribution of a set of values is obtained by indicating the frequency for each value or by diving the range of values in buckets and counting the number of elements in each bucket.

To represent a dataset by its quartiles we use a boxplot[6] representation (Fig. 7.7) in this way we graphically know where is the center of the dataset.

7.2.2 Inferential statistics

The parameters used in descriptive statistics are random variables which values is determined by the causality of the sample. Typically samples have a Gaussian distribution so we can perform several estimations. In inferential statistics we don't have all the data (as in descriptive statistics) so our parameters refer only to the sample's elements and we want to use them, along with the distribution of the dataset to infer information on the whole dataset, X.

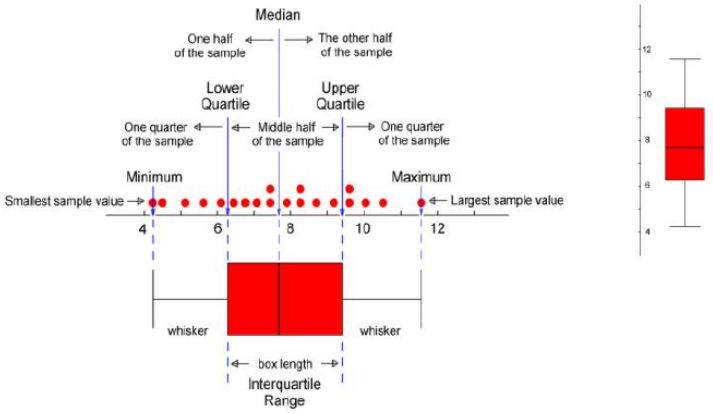


Figure 7.7: Boxplot representing the frequency distribution of dataset

Confidence interval.

Definition. *Confidence interval: Probability that the mean of a population X is in a finite interval centered on the mean of the sample of n elements of the population.*

The size of the intervals determines the probability of error, which proportional to the standard deviation an inversely proportional to the size of the sample. It allow to estimate a population parameter, by using an interval which is likely to include that parameter; the confidence level will determine the reliability of the estimate. To increase the confidence level we must widen the interval, to increase the number of possible values in the sample that are expected to include the true population parameter (Fig. 7.8). Let x be the parameter value, the confidence interval then is $[x - d, x + d]$ with confidence level p.

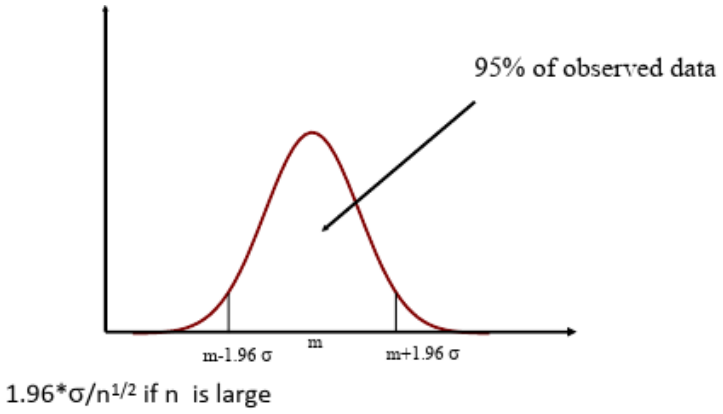


Figure 7.8: An example of a confidence interval

Hypothesis verification. Often we need to compare different repeated measures, to do so we perform appropriate statistic tests.

Definition. *Null Hypothesis H_0 [7]: General statement that says that all true means are equal and the differences are random. This hypothesis is considered true until evidence indicates otherwise.*

Definition. *p-value [8]: Probability that a random variable has a value greater that a given threshold if the null hypothesis is true. The value of the probability is computed as $\int_x^{+\infty} D(x)dx$ where x is the measured value and D is the distribution associated with the chosen random variable.*

The verification of H_0 uses a fixed error probability, α. We can test if two samples come from the same population. To so we use H_0 on one or more random variables, we also define two hypotheses:

H_0 : samples come from the same population

H_1**:** samples come from different populations

We need to define a formula which captures the difference between these data by generating a random variable and use the p-value to check which hypothesis is true. The value of p is the probability to reject H_0 when is true. We proceed like this:

- $p \geq \alpha \Rightarrow$ we accept H_0
- $p \leq \alpha \Rightarrow$ we accept H_1

The value of α is fixed so we need to tune it according to how critical our decision is, because critical decisions use little value of α.

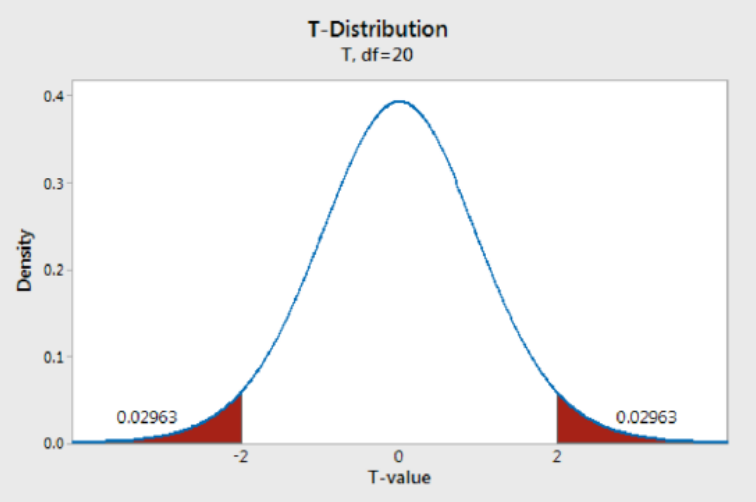


Figure 7.9: Distribution of the random variable used for computing the p-value (in red we have the reject zones)

Student's t-test.

Compares the mean values of two samples, with n elements each.

$$t = \frac{\mu_a - \mu_b}{\sqrt{\frac{\sigma_a^2 + \sigma_b^2}{n}}}$$

Assuming that a and b come from the same population we can compute the probability density of t and the probability of t being equal or greater than a value. This is expressed by a table that indicates the probability that $t \geq \alpha$, according to the degree of freedom (sample dimension); given n elements and the mean we have n-1 degree of freedom for a single sample and the overall number of degree of freedom is 2(n-1); also when reading the table we must remember that the α value given is for a single tail of the distribution (e.g. if $\alpha = 5\%$ the we must look an the table where $\alpha = 2.5\%$). To determine if we can accept H_0 we need to compare the computed t-value with the critical t-value given by the table with the corresponding value of α and the same degree of freedom; if our value is less than the t-critical value the we can accept H_0, otherwise we need to reject it.

Probability of error.

- H_0 is true:
  - H_0 is rejected with probability α and we have a false positive
  - H_0 is accepted with probability $1 - \alpha$
- H_0 is false:
  - H_0 is rejected with probability $1 - \beta$
  - H_0 is accepted with probability β and we have a false negative

  However β cannot be computed since it depends on the mean of the dataset, which is not known.

Analysis of variance (ANOVA).

Is a technique similar to the t-test to evaluate several (≥ 2) samples at once [9]. We assume that all the samples are distributed by a Gaussian distribution. The hypothesis we have are the following:

H_0 : all samples have the same mean

H_1**:** the are at least two different means between all the samples

Sum of squares. The first step in the ANOVA test is the computation of the variance within each sample, SS_W and the variance between all the samples SS_b. Let I be the number of samples, J be the number of element in each samples, Y_{ij} the jth element of the Ith sample and μ be the mean of all the elements in all the samples.

$$SS_w = J \cdot \sum_{i=1}^{I} \sum_{j=1}^{J} (Y_{ij} - \mu_i)^2$$

$$SS_b = J \cdot \sum_{i=1}^{I} (\mu_i - \mu)^2$$

Fisher test. After having computed the variance parameter we can execute the Fisher test, by computing the F value:

$$F = \frac{\frac{SS_b}{I-1}}{\frac{SS_w}{I \cdot (J-1)}}$$

and proceed to the validation of the F value with the F-critical value as it is done with the t-test. If the ANOVA test leads us to reject the H_0 hypothesis then we must test each couple of samples to find out in which distribution each sample is.

7.2.3 Examples of application

We can use the t-test and ANOVA test to prove general statements in software production and testing. For example we could use the ANOVA test to see if the ratio between tested KLOC and written KLOC increases then the defect rate in the first year decreases, on several packages at once.

Chapter 8

Docker Basics

8.1 VMs, Containers and Docker

Virtual Machines vs Containers.

Definition. *Containers have similar resource isolation and allocation benefits as virtual machines but a different architectural approach allows them to be much more portable and efficient.*

Containers are capable to run application isolating them from the OS and the hardware; they are different than a virtual machine which, other than the application and the necessary libraries, includes also an entire operating system. (Fig. 8.1) Containers instead, share the kernel with other containers and run in userspace isolted form the host OS and other containers, unless otherwise specified; they are not tied to a specific infrastructure and the images are based on layered file systems, to share common files.

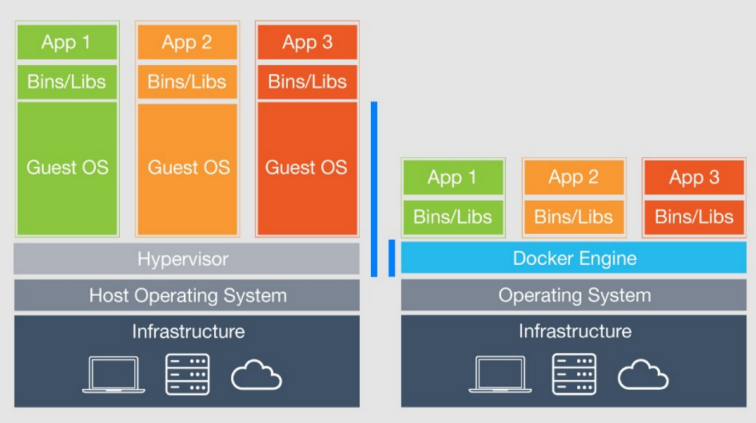


Figure 8.1: Comparison between several VMs (left) and several containers (right) needed resources

Containers. Containers are not a new technology, since they are supported since the Linux kernel v3.8 and in Linux the support was given by LXC, with

cgroups and namespaces, which give support to manage containers, advanced networking and storage, but also providing container templates. The eveolution of LXC project is the Docker project, which was initially based on LXC and features single application containers.

LXC vs Docker. They both use kernel namespaces to provide containers that live in userspace, other difference between them is that an LXC container has an init process, so it can run mulitple processes, instead a docker container can run only a single process, not having an init.

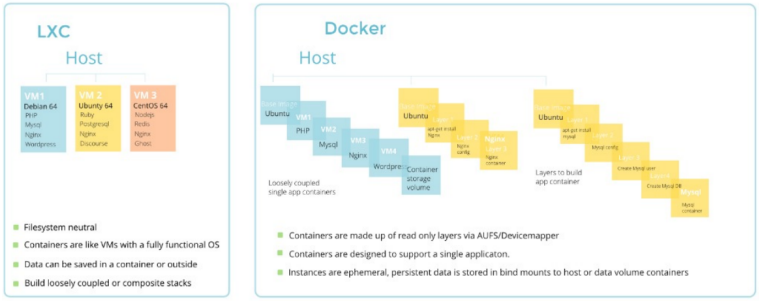


Figure 8.2: Comparison between docker and LXC containers.

Docker. Docker [10] allows you to package an application with all of its dependencies into a standardized unit for software development, wrapped up in a complete file system that contains everything needed to run the application, so it guaranteed that the application will always run in the same ambient, even when the hardware and the OS changes. Containers use the docker abstracted system resources and have different layers, to share common files with other containers (Fig. 8.3). Typically, when designing an application or service to use Docker, it

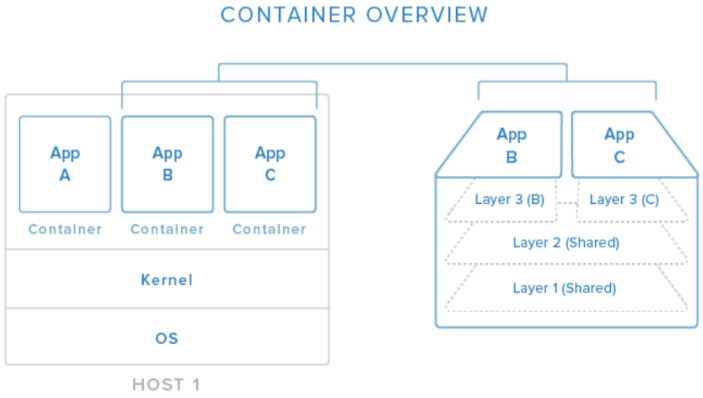


Figure 8.3: Layered containers resource sharing example.

works best to break out functionality into individual containers, a design recently known as micro-service architecture. This gives you the ability to easily scale or

update components independently in the future. This makes docker interesting for developing and deployment because offers several advantages:

- Containers are lighter than VMs
- Portability of the application on any host running Docker
- The container is isolated so the application is accessible only by the exposed interface, which need to be well-documented, making interactions predictable.

8.2 Docker Engine

Docker architecture. The docker engine is a client-server program composed by a daemon with REST APIs and a command line interface which is used to give commands to the daemon through the exposed APIs. (Fig. 8.4) The docker daemon can run either on the same system or on a remote system and it builds, distributes and runs the containers.

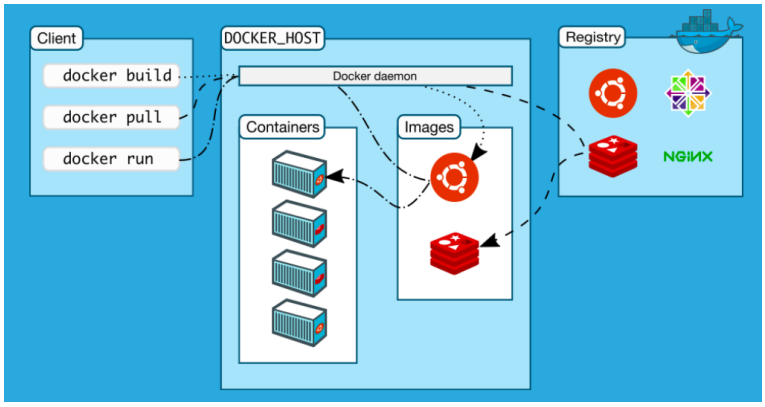


Figure 8.4: Docker engine architecture.

8.3 Custom Images

To build custom images [11] we can create a container and use it to build our application manually or automatize the installation of the application and its requirements with a dockerfile.

Dockerfile. A Dockerfile defines what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you need to map ports to the outside world, and be specific about what files you want to "copy in" to that environment. However, after doing that, you can expect that the build of your app defined in this Dockerfile behaves exactly the same wherever it runs. Some of the common commands that we can use in a Dockerfile are:

FROM: sets the parent image of the current image

WORKDIR: sets the working directory

COPY: copies file and directories into the container

RUN: runs console commands

EXPOSE: opens ports from container, to reach if from the outside

ENV: defines environment variables

CMD: commands to be run when the container starts

8.4 Docker Compose

Compose [12] is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration. Using Compose is basically a three-step process:

1. Define your app's environment with a Docker file so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Lastly, run docker-compose up and Compose will start and run your entire app.

Chapter 9

Software metrics

To measure software we can use different metrics:

- On the code (Lines of code, McCabe index)
- On the requirements (Transactional)
- Indirect (Service levels, user's opinions)

9.1 Lines of Code

These metrics are used to evaluate the dimension of softwar products in terms of lines of code, they can be:

- Internal errors/KLOC
- external errors/KLOC/year
- Documentation pages/KLOC
- LOC/person/month
- Errors/person/month
- cost per LOC
- cost per documentation page

[LOC: Line of code,1 KLOC = 1000 LOC] These metric seems easy to compute but it tightly coupled with the programming language used, since many lanaguages support code generation or require different amounts of LOCs to implement the same functionalities.

9.2 Function Points

This is an alternative to the LOC metric which tries to overcome its weaknesses, the first proposal was called Function Points (FP) and was an empirical formula based on weighted simple functionalities. This proposal has received several extensions. It base idea it not to measure the lines of code that are written but to measure the offered functionalities, which will be converted into the approriate number of LOCs for the used language. (Fig. 9.1)

Programming Language	LOC/FP
Assembler	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
OO (C++ / JAVA)	30
4GL	20
Generatore di codice	15
Foglio elettronico	6
Linguaggio grafico/visuale	4

Figure 9.1: Conversion table from FP to LOCs, according to the used language

9.2.1 Data functionalities

We consider are data functionalities:

Internal logical file (ILF): User-identifiable group of logically related data or control information maintained within the boundary of the application.

External Interface File (EIF): User-identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted. An EIF counted for an application must be in an ILF in another application.

To evaluate the ILFs and EIFs complexity we use two parameters in conjunction with the functionalities we use for FP:

Data Element Type (DET): User identifiable single field within a ILF / EIF (equivalent to a record in a table)

Record Element Type (RET): User identifiable group of fields within a ILF / EIF (equivalent to a table)

We can express the ILF/ELF complexity in base of the number of RETs and DETs in the file. (Fig. 9.2)

9.2.2 Transactions

We also consider transactions that happens using the above stored data (Fig. 9.3):

Ret/Det	1-19 Det	20-50 Det	51+ Det
1 Ret	Low (7/5)	Low (7/5)	Medium (10/7)
2-5 Ret	Low (7/5)	Medium (10/7)	High (15/10)
6+ Ret	Medium (10/7)	High (15/10)	High (15/10)

Figure 9.2: ILF/ELF complexity based on RETs and DETs

External Input (EI): Elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.

External Output (EO): An elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic. The processing logic must contain at least one mathematical formula or calculation, create derived data, maintain one or more ILFs or alter the behavior of the system.

External Inquiry (EQ): An elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF of EIF. The processing logic contains no mathematical formulas or calculations, and does not create derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.

Actions	EI	EO	EQ
1) Validate	can	can	can
2) Math calculations	can	must*	cannot
3) Select data using specific criteria	can	can	can
4) Evaluate Boolean conditions	can	can	can
5) Update one or more ILF	must* / p.goal	must*	cannot
6) Read one or more ILF /EIF	can	can	must
7) Get some control data	can	can	must
8) Compute new data	can	must*	cannot
9) Change the system behavior	must*/p.goal	must*	cannot
10) Find data and present them outside the application boundary	can	must/p.goal	must/p.goal
11) Acquire (control) data produced outside the application boundary	must	can	can
12) Manipulate data (e.g., sort)	can	can	can

Figure 9.3: Transaction indentification

For all these transactions we have two components:

- the file type referenced (FTR) which is the read/written ILF or ELF
- the DETs involved in the transactions

We use these two components to evaluate the complexity of our transactions (Fig. 9.4 and 9.5)

FTR / DET	1-4 DET	5-15 DET	16+ DET
0 – 1 FTR	Low (3)	Low (3)	Medium (4)
2 FTR	Low (3)	Medium (4)	High (6)
3+ FTR	Medium (4)	High (6)	High (6)

Figure 9.4: Evaluation table for EI transactions

FTR / DET	1-5 DET	6/19 DET	20+ DET
0 – 1 FTR	Low (4/3)	Low (4/3)	Medium (5/4)
2-3 FTR	Low (4/3)	Medium (5/4)	High (7/6)
4+ FTR	Medium (5/4)	High (7/6)	High (7/6)

Figure 9.5: Evaluation table for EO and EQ transactions

9.2.3 FP computation

Each functionality, data and transactions, have three possibles weights different for each of the above categories. In the 1984 proposal the points associated to a project where computed in this way[1]:

$$FP = [\sum_{i \in \{ILF,EIF,EI,EO,EQ\}} (\sum_{j \in func_i} count_j \cdot weight_j)] \cdot (0.65 + 0.01 \cdot \sum_{k}^{14} F_k)$$

Where $func_j$ is a functionality in the set of the five general functionalities we previously defined and $weight_j$ is the associated weight; F_k are the factors that can influence the projects, like if the system is distibuted, the language used etc. which are represented by 14 indicators, which can have a value from 0 to 5, adjusting the value up to 35%. Summing up FP usage we have the following advantages:

- Widely used and accepted (standards, active organizations)
- Certified personnel available
- Objective calculation
- UFP independent of technology
- Can be used early in development process
- Equally accurate as SLOC

[1] To find exmples on how FP and LOCs are computed see slides of lesson 7 and 8

And the following disadvantages:

- Semantic difficulty - “legacy” terminology difficult for teaching, and FP are in themselves hard to grasp and compare
- Incompleteness – internal functionality? Stored data size vs. complex processing?
- Lack of automatic count
- Different versions

Chapter 10

Effort Estimation

Effort estimation is the process which allows to determine the cost, effort and time needed for as software project given its requirements. To proceed in effort estimation the step performed are:

1. Requirements collection
2. Function points estimation
3. Lines of code estimation
4. Time and effort estimation
5. Cost estimation

In general a project with many function points is likely to be abandoned or overdue, in opposition to a project that has fewer function points which will be likely to be completed in time.

10.1 Constructive Cost Model

The CoCoMo [13] is a cost model based on the waterfall development process that relies on statistics to estimate effort and cost for a software project, is composed on three models which have slightly different formulas to estimate effort according to the project phase in which effort needs to be estimated:

- Analysis and planning
- Design
- Development
- Integration and test

In detail the estimation regards several factors:

Effort, Cost (M): man time require to develop the project [man-years,man-weeks..]

Delivery time (T): Time required to deliver working software (with work palatalization [years, weeks..]

Manpower: Number of people working during the project $[\frac{M}{T}]$

These parameters are adjusted according to the context in which the software is developed, which is represented by other parameters that are measured in scale of six values.

10.1.1 1981 model

In the 1981 there are three models to be used according to the difficulty of the project, and the project can be estimated only in its dimension or according to the correction factors described above (represented here as c_i). If we evaluate only the project dimension we have the following formulas with S_k as the estimated lines of code, assuming that the requirements do not change:

Simple: $M = 2.4 \cdot S_k^{1.05}$ and $T = 2.5 \cdot M^{0.38}$

Intermediate: $M = 3.0 \cdot S_k^{1.12}$ and $T = 2.5 \cdot M^{0.35}$

Complex: $M = 3.6 \cdot S_k^{1.2}$ and $T = 2.5 \cdot M^{0.32}$

If we consider the global correction coefficient we obtain:

Simple: $M = 3.2 \cdot S_k^{1.05} \cdot \prod_1^{15} c_i$ and $T = 2.5 \cdot M^{0.38}$

Intermediate: $M = 3.0 \cdot S_k^{1.12} \cdot \prod_1^{15} c_i$ and $T = 2.5 \cdot M^{0.35}$

Complex: $M = 2.8 \cdot S_k^{1.2} \cdot \prod_1^{15} c_i$ and $T = 2.5 \cdot M^{0.32}$

In this analysis the time of planning and analysis of the requirements is not considered, we consider a man-month (MM), which is the unit of measure of M to be composed of 152 hours (19 days of 8 hours), T instead is represented in months. As a general trend, the more lines of code in the project, the more time is spent in the testing and coding phases. To represent the phases duration in the project we use a Gantt chart[14]. As we said before the 1981 model uses several assumptions:

- Waterfall model
- Stable requirements
- Adequate personnel
- Project management
- less than 28% of error in 68% of estimates

10.1.2 CoCoMo II

The second version of the CoCoMo [15] model was born to address the limitation of the 1981 version and to consider software reuse, different lifecycle and different levels of precision in the estimations. This new version consists of two models:

Early Design model: there is a low level of detail in this model and few parameters (7), it's used in the initial phases of the project and uses function points to estimate the effort.

Post-Architecture model: has more detail and parameters (17) than the early model and evaluates the effort taking into account function points and reuse; it's used for the development and maintenance phases.

These two models share 5 scaling factors for computing the exponents factors.

CoCoMo II formulas. Let $SCED$ be the required development schedule, n be either $6 + SCED$ or $16 + SCED$, EM_i be the effort multipliers that adjust the model according to the environment and A,B,C,D constants. We compute the person months, PM, in the following way:

$$PM = A \cdot S^E \cdot \prod_i^n EM_i$$

Where S is the estimated size of the project in KLOC.

The PM computation requires the computation of the economy/diseconomy of scales, E (in CoCoMo 81 there were only diseconomies), where SF_j are the five scale factors that CoCoMo II uses:

$$E = B + 0.01 \cdot \sum_{j=1}^{5} SF_j$$

The development time,T, is computed as follows:

$$T = C \cdot PM^F \cdot SCED/100$$

$$F = D + 0.2(E - B)$$

Scale Factors. The Scale factors are (Fig. 10.1):

Precendentedness (PREC): familiarity with the work, given by past similar works, it's intrinsic to the project and uncontrollable. (Fig. 10.2)

Development Flexibility (FLEX): Flexibility and relaxation during work, intrinsic to the project and uncontrollable (Fig. 10.3)

Architecture / Risk Resolution (RESL): Combines design thoroughness and risk elimination strategies included in the project (Fig. 10.4)

Team Cohesion (TEAM): Sources of project turbulence given by difficulties int synchronizing the stakeholders (difficulties created by people involved in the project) (Fig. 10.5)

Process Maturity (PMAT): Process maturity level measured according to the CMM-levels (Fig. 10.6). If CMM-levels are not available the EPML (Estimated Process Maturity Level) is computed as the percentage of compliance to the 18 Key Process Area goals defined by CMM, according to the following formula: $EPML = 5 - [(\sum_{i=1}^{n} \frac{KPA\%_i}{100}) \cdot \frac{5}{18}]$.

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
SF_i:	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_i:	5.07	4.05	3.04	2.03	1.01	0.00
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
SF_i:	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_i:	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	The estimated Equivalent Process Maturity Level (EPML) or					
	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5
SF_i:	7.80	6.24	4.68	3.12	1.56	0.00

Figure 10.1: Scale Factor Values, SF_j, for COCOMO II Models

Effort multipliers. We have 17 effort multipliers that can be used in the second model an only 7 of them can be used in the first model, these multipliers are divided in different categories (Fig. 10.7):

Product :

RELY: Required product reliability

DATA: Database size

CPLX: Product complexity

RUSE: Intended reuse of software models

DOCU: Level of required documentation

System :

TIME: Execution time constraints

STOR: Main storage constraint

PVOL: Platform volatility (change frequency)

Personal :

ACAP: Analyst capability

PCAP: Programmer capability

APEX: Application experience

PLEX: Platform experience

LTEX: Language and tool experience

PCON: Personnel continuity

Feature	Very Low	Nominal / High	Extra High
Precedentedness			
Organizational understanding of product objectives	General	Considerable	Thorough
Experience in working with related software systems	Moderate	Considerable	Extensive
Concurrent development of associated new hardware and operational procedures	Extensive	Moderate	Some
Need for innovative data processing architectures, algorithms	Considerable	Some	Minimal

Figure 10.2: Precedentedness Rating Levels

Development Flexibility			
Need for software conformance with pre-established requirements	Full	Considerable	Basic
Need for software conformance with external interface specifications	Full	Considerable	Basic
Premium on early completion	High	Medium	Low

Figure 10.3: Development Flexibility Rating Levels

	Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Product Design Review	Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR.	None	Little	Some	Generally	Mostly	Fully
	Schedule, budget, and internal milestones through PDR compatible with Risk Management Plan	None	Little	Some	Generally	Mostly	Fully
	Percent of development schedule devoted to establishing architecture, given general product objectives	5	10	17	25	33	40
	Percent of required top software architects available to project	20	40	60	80	100	120
	Tool support available for resolving risk items, developing and verifying architectural specs	None	Little	Some	Good	Strong	Full
Component Off The Shelf	Level of uncertainty in Key architecture drivers: mission, user interface, COTS, hardware, technology, performance.	Extreme	Significant	Consider-able	Some	Little	Very Little
	Number and criticality of risk items	> 10 Critical	5-10 Critical	2-4 Critical	1 Critical	> 5Non-Critical	< 5 Non-Critical

Figure 10.4: RESL Rating Levels

Characteristic	Very Low	Low	Nominal	High	Very High	Extra HIgh
Consistency of stakeholder objectives and cultures	Little	Some	Basic	Consider-able	Strong	Full
Ability, willingness of stakeholders to accommodate other stakeholders' objectives	Little	Some	Basic	Consider-able	Strong	Full
Experience of stakeholders in operating as a team	None	Little	Little	Basic	Consider-able	Extensive
Stakeholder teambuilding to achieve shared vision and commitments	None	Little	Little	Basic	Consider-able	Extensive

Figure 10.5: TEAM Rating Components

Project :

SITE: Multisite development

TOOL: Use of software tools

SCED: Schedule constraints

All these multipliers make both models very variable when determining the effort needed to develop the project, even if the models have different multipliers (Fig. 10.8).

10.1.3 Software Reuse

We need to model software reuse since it is not a trivial process, due to the fact that the code we want to reuse can be subject to some modifications and has a certain level of familiarity, readability and documentation. To take into account all these factors we use a non linear module which models the effort to adapt a an existing module as the effort to develop a new module, measuring the equivalent lines of code (ESLOC). This model is based on two aspects:

- The complexity to adapt software which is derived from:

  Software understanding (SU): how the software is easy to read, understand and modify to be used in the new project in terms of documentation, readability and modularity of the code (from low to high: $[50, 40, 30, 20, 10]$)

  Assessment and Assimilation (AA): if the code can be useful for the application and how it documentation can be integrated with the actual product through test, evaluation to process and documentation that needs to be written. (from none to extensive: $[0, 2, 4, 6, 8]$)

  Programmer Unfamiliarity(UNFM): of the software to be integrated (from familiar to unfamiliar: $[0, 0.2, 0.4, 0.6, 0.8, 1]$)

- The percentage of modification, the Adapting Adjusting Factor (AAF)

  DM: percentage of modified design

  CM: percentage of modified code

  IM: percentage of the modification of the integration effort without reusing code

We can compute the Equivalent SLOC in two ways, depending on the adapting adjusting factor (AAF).

$$AAF = (0.4 \cdot DM) + (0.3 \cdot CM) + (0.3 \cdot IM)$$

$$AAM = \begin{cases} \frac{[AA+AAF\cdot(1+(0.02\cdot SU\cdot UNFM))]}{100} & \text{if } AAF \leq 50 \\ \frac{AA+AAF+(SU\cdot UNFM)}{100} & \text{if } AAF > 50 \end{cases}$$

After having computed both the Adaptation adjustment factor and modifier, we have that the estimated KLOCs are:

$$EKLOC = AKLOC \cdot (1 - \frac{AT}{100}) \cdot AAM$$

Where $AKLOC$ are the adapted lines of code that are modified to be of use in the actual project and AT is the percentage of code that is re-engineered by automatic translation. With this parameter we can compute the relative cost and modification size required to reuse a piece of code in a project (Fig. 10.9)

10.1.4 Backfiring

After we have computed our function points we can compute with them the SLOC, in order to do that we can use tables, which indicates the backfiring (the average correspondence between the lines of code SLOC and the function poins). The backfiring tables can be consulted from page 59 of CoCoMo slides.
The LOC estimation can be done using the given data as follows:

- 51 FP
- C Language
- Backfiring for C language = 128
- LOC = $51 \times 128 = 6528 = 6.528$ KLOC

10.2 Proposed methodology

With the CoCoMo II model we can estimate with a good approximation the time necessary to the delivery of the software and the effort that we need to complete the project with a certain delivery time, so we can estimate the cost of the project, tanking into account the cost that we sustain if we write new code and the cost that we have when we reuse some modules. (Fig. 10.10)

PMAT Rating	Maturity Level	EPML
Very Low	CMM Level 1 (lower half)	0
Low	CMM Level 1 (upper half)	1
Nominal	CMM Level 2	2
High	CMM Level 3	3
Very High	CMM Level 4	4
Extra High	CMM Level 5	5

Figure 10.6: PMAT Ratings for Estimated Process Maturity Level (EPML)

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconve-nience	low, easily recoverable losses	Moderate, eas-ily recoverable losses	high financial loss	risk to human life	
DATA		DB bytes/Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
CPLX	see Table II-15					
RUSE		none	Across project	across program	across product line	across multi-ple product lines
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very exces sive for life-cycle needs	
TIME			50% use of available exe cution time	70%	85%	95%
STOR			50% use of available stor age	70%	85%	95%
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
AEXP	≤ 2 months	6 months	1 year	3 years	6 years	
PEXP	≤ 2 months	6 months	1 year	3 years	6 year	
LTEX	≤ 2 months	6 months	1 year	3 years	6 year	
TOOL	edit, code, debug	simple, fron tend, backend CASE, little integration	basic lifecycle tools, moder ately inte grated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive life cycle tools, well inte grated with processes, methods, reuse	
SITE: Colloc ation	International	Multi-city and Multi-com pany	Multi-city or Multi-com pany	Same city or metro. area	Same building or complex	Fully collo cated
SITE: Comm unicati ons	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communica tion.	Wideband elect. comm, occasional video conf.	Interactive multimedia
SCED	75% of nomi nal	85%	100%	130%	160%	

Figure 10.7: Summary of all the Effort multipliers

Early Design Cost Driver	Counterpart Combined Post-Architecture Cost Drivers
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	AEXP, PEXP, LTEX
FCIL	TOOL, SITE
SCED	SCED

Affects both M and T

Figure 10.8: Correspondence between models multipliers

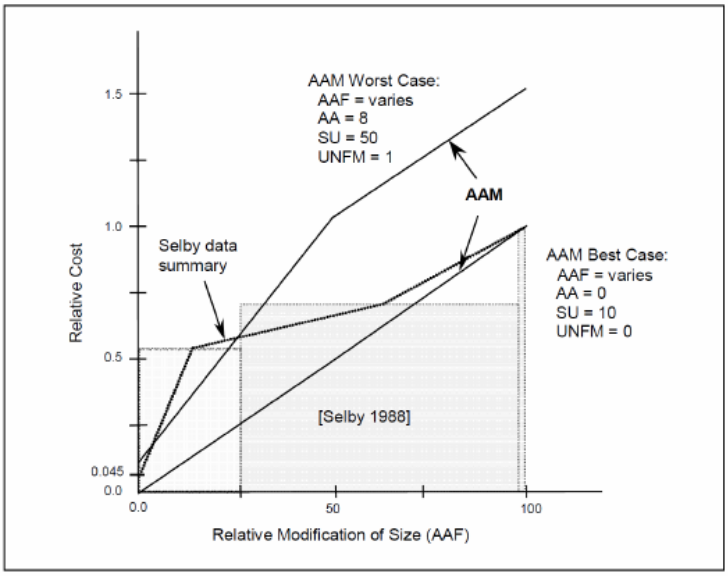


Figure 10.9: AAM values in best and worst cases

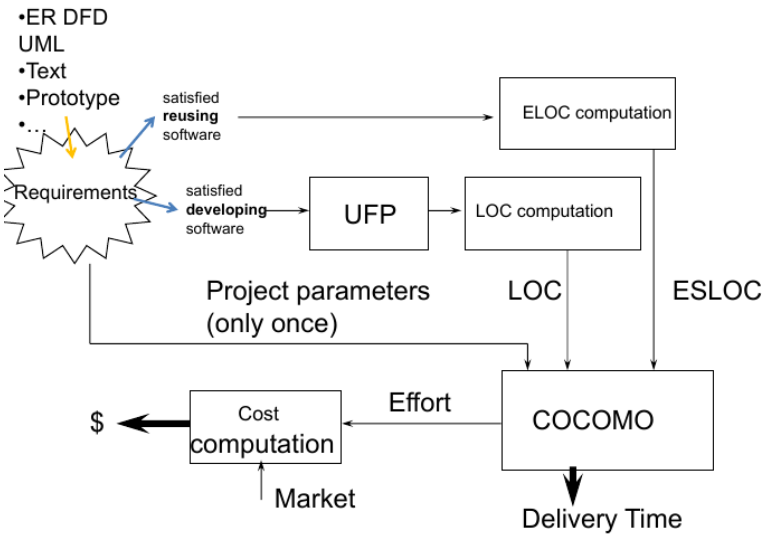


Figure 10.10: Cost and effort estimation methodology

Bibliography

[1] "Miroservices." [Online]. Available: https://en.wikipedia.org/wiki/Microservices

[2] "Why use a microservices approach to building applications?" [Online]. Available: https://docs.microsoft.com/en-gb/azure/service-fabric/service-fabric-overview-microservices

[3] "What are microservices?" [Online]. Available: https://aws.amazon.com/microservices/

[4] C. Richardson, "Microservices." [Online]. Available: https://microservices.io/

[5] "Systematic error / random error: Definition and examples." [Online]. Available: https://www.statisticshowto.datasciencecentral.com/systematic-error-random-error/

[6] "Box plot." [Online]. Available: https://en.wikipedia.org/wiki/Box_plot

[7] "Null hypothesis." [Online]. Available: https://en.wikipedia.org/wiki/Null_hypothesis

[8] "p-value." [Online]. Available: https://en.wikipedia.org/wiki/P-value

[9] "Analysis of variance." [Online]. Available: https://en.wikipedia.org/wiki/Analysis_of_variance

[10] "Docker." [Online]. Available: https://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf

[11] "Getting started with docker." [Online]. Available: https://scotch.io/tutorials/getting-started-with-docker

[12] "Getting started with docker compose." [Online]. Available: https://docs.docker.com/compose/gettingstarted/

[13] "Software engineering | cocomo model." [Online]. Available: https://www.geeksforgeeks.org/?p=193526

[14] "Gantt chart." [Online]. Available: https://en.wikipedia.org/wiki/Gantt_chart

[15] "Cocomo ii model definition manual." [Online]. Available: https://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf