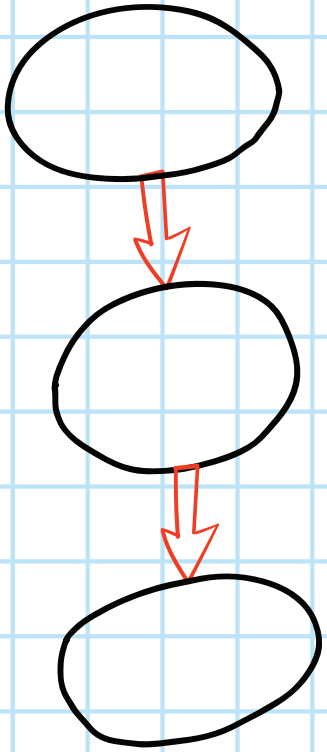
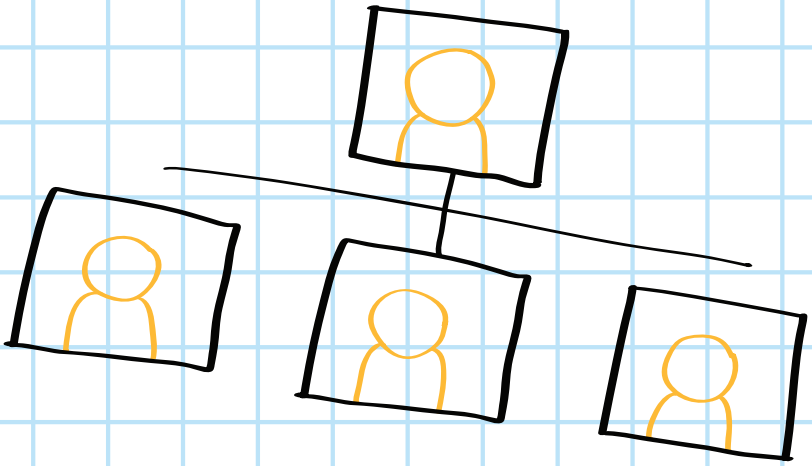
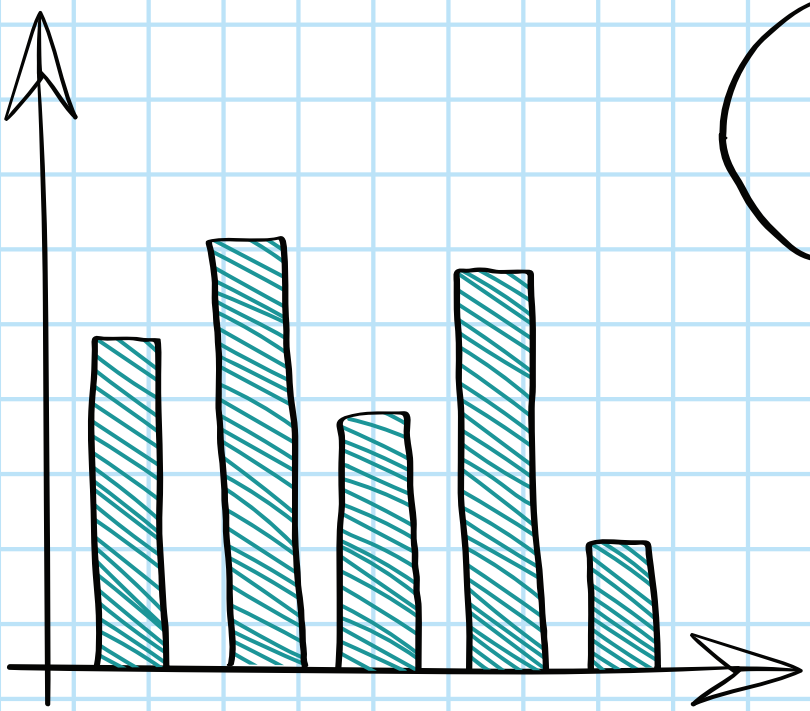
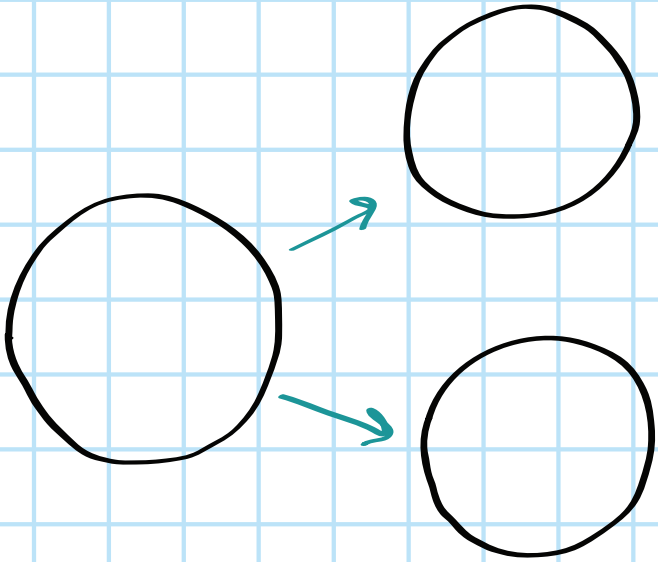
Software Engineering





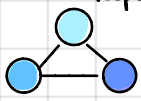
Edoardo Puglesi





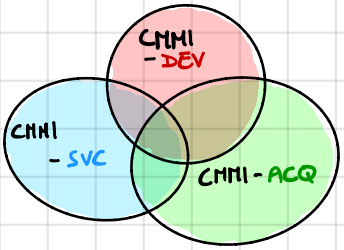
SOFTWARE DEVELOPMENT PROCESS

Hardware cost << programmer cost → make programmer efficient (not the program)
The better the quality of development process the better the quality of software → creation of standards for software development processes (ISO).



Process: sequence of steps

Process Model: structured collection of practices proven to be effective by experience.
Capability Maturity Model Integration (CMMI): the process model is continously improved to grant best model to organizations



Constellations:
DEV: guidance for managing, measuring and monitoring development process
SVC: guidance for delivering services (internally or not)
ACQ: guidance for applying CMMI best practices in an acquiring organization

CMMI is divided in levels (0 to 5) according to software maturity
CMMI models contain multiple Process Area: Process Management, Project Management, Engineering and Support are the main process area categories

ISO 12207

Standard for software lifecycle processes. Its main goal is to provide common language for stakeholders
Its based on activities that transform input to output: 5 primary lifecycles related to main agents involved (buyers, suppliers, developers, maintainers, operators, managers, technicians), 8 supporting lifecycles, 4 organization lifecycles.

- Modularity
- Responsability: establish a responsability for each process facilitating standard application.

Note: model/standard just say which activities must be done, not how!

ISO 9000

Standard for quality management systems, certify the quality of management process not of software itself. ISO 9000 contains multiple standards eg ISO 9001 specify requirements of a quality management system.
ISO 9000 can be divided in 5 blocks: quality management system, management responsability, Resource management, Product/service realization, Measurement analysis and improvement.
Quality requirements: set of process requirement and resources that constitute the Quality Manual (QM) of the organization, adapted to specific projects generating several Quality Policies (QP). Both together are the documentation needed by ISO 9001 certification

There are different process model according to activities and documentation produced. The monitoring is based on the explicit definition of activities to be performed and documentation to be produced (to evaluate quality).

① Waterfall model: separate and distinct phases of specification and development:
- requirements analysis and definition
- system and software design
- implementation and unit testing
- integration and system testing
- operation and maintenance

→ Each phase must be completed to move to the next one

CONS: can't do changes after the process started, end user has not an overall vision and must wait termination to judge and discover errors and at some time programmers must wait analysis phase to start working.

② Evolutionary development: specification, development and validation interleaved.

③ Component-based software engineering: part of the system is assembled from existing components

④ Formal model: requirements expressed in formal language.

⑤ Process Iteration: systems requirements ALWAYS evolve during the project
- ↳ prototypal model: customer interaction → prototype → test with costumer → repeat
- ↳ incremental model: analysis → design → implementation → test → version #n → repeat

The difference is that prototype implements only few basic feature (UI) while versions are fully working.

⑥ Incremental Development: define requirements → assign req. to increments → design architecture →
→ develop system increment → validate increment → integrate increment → validate system → final system
↑ incomplete

The software is delivered partially prioritizing important increments.
PROS: lower risk of overall failure, early increments act as prototype

⑦ Extreme programming (Agile family): based on development and delivery of very small increments of functionality. More programming, less paper, team of two.

⑧ Spiral development: each loop represents a phase in the process (decided on priority), evaluation at the end of each loop (risk analysis). Develop → review → plan next phase

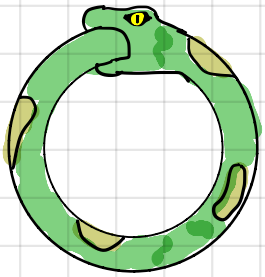
PROCESS ACTIVITIES

- Software specification
- Software design and implementation
- Software validation
- Software evolution (maintenance)

SOFTWARE SPECIFICATION

Establish what services are required and the constraints on system operation and development.
The first loop of spiral development, for example, is based only on this.
Functional and not-functional (just for quality) requirements.

- Feasibility study: is the project realizable in practice?
- Requirements elicitation and analysis: ask requirements to customers and analyse them
- Requirements specification: write requirements in a formal way eg UML schema
- Requirements validation



SOFTWARE DESIGN & IMPLEMENTATION

Convert specification into executable system.
Design = architecture, interface (API / User), components, Data structure and algorithm
Implementation = programming to translate design to program + debugging

SOFWARE VALIDATION

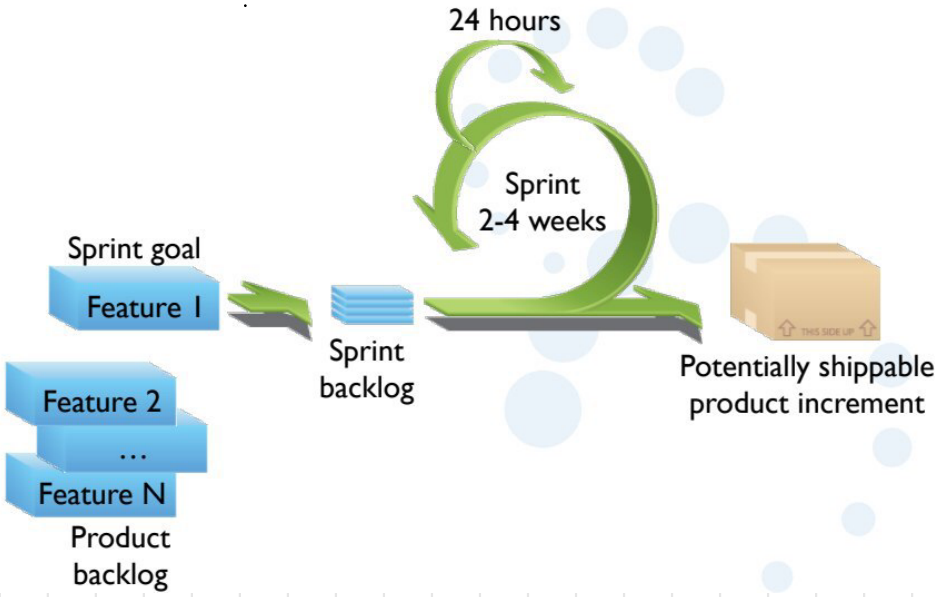
Verification & Validation (V&V): check if conform to specification (verification) and meets customer's requirements (validation). The second one is the most difficult and need customer's partecipation.
System testing and acceptance testing.

SOFTWARE EVOLUTION

Software can be flexible and can change according for example to new requirements. Rigth now development = evolution since fewer systems are completely new.

SCRUM (part of Agile)

- Self organizing teams
- Work divided in 2-4 week "sprints" (iterations)
- Requirement listed as "product backlog" = list of features to be implemented (ordered)
- Can use whatever technology the developer wants.
- Generative rules



Note: instead of a doing all of one thing at time, scrum does a little of everything all time. (Requirements, code, design and Test). No changes during sprints.

ROLES

Product owner (customer), Scrum Master (Project manager) and Teams (5-9 persons / indipendent)

CERIMONIES

- Sprint Planning Meeting: at start of every sprint, needed to organize the next sprint in term of goal and backlog. Teams select items from product backlog (user story) they can commit to completing and estimate the needed hours.
- Daily Scrum: daily meetup standing up to see if there are issues (not problem solving)
  What did yesterday? What today? What's next?
- Sprint Review: teams present what they did in the sprint
- Sprint Retrospective: periodically take a look at what is / is not working. What to continue, start, stop.

ARTIFACTS

- Product Backlog: list of user stories
- Sprint Goal: focus of the sprint
- Sprint Backlog: division of backlog on the whole sprint. Everyone can add/remove backlog from it.
- Sprint Burndown Chart: show work completed and not. One for each developer or work item.
  (x = days y = remaining tasks)

BEHAVIOUR-DRIVEN DESIGN (BDD - user stories)

Requirements are written as user stories (light description) that concentrates on behaviour of app.

FORMAT:

> As a [subject]
> So that [goal to achieve]
> I want to [how to achieve it]

One for each action / user
Easy to organize
Different stakeholders (subject) can see the same behaviour differently

Every story has a difficulty point → team productivity is measured in points / week
Stories can be subdivided in less complex "stories" called epics.

They must be SMART

- Specific
- Measurable

→ Testable, anti eg. "UI should be user friendly" - not measurable

- Achievable: complete in 1 iteration, split if necessary.
- Relevant: must be relevant in term of business (includes make the product remarkable)

In addition to user stories BDD makes use of LO-FI sketches (eg UI)
They are simple pen-paper sketches (or online drawings)
From that we create the storyboard of how the interface evolve interacting with user.

DISTRIBUTED PROGRAMMING

LAYERS: Presentation Layer, Application Logic Layer (what the systems does) and Resource Management Layer (deals with data for logic layer)

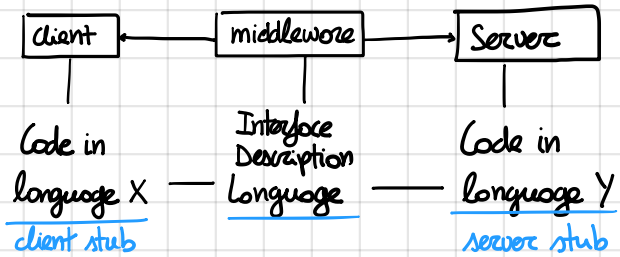
1-Tier Architecture: one single physical machine in which all layers are deployed.
2-Tier Architecture: client "host" the presentation layer while server host the other two. Typical client-server architecture.

Interfaces & "Services": services are used to invoke system function while Interface are basycally the API to call them.

3-Tier Architecture: Each layer is by itself. Middleware is a level of indirection between client and other layers.
N-Tier Architecture: layers are more complex and divided in modules eg Presentation layer = Web server + HTML filter + Web Browser (Client)
This is the scenario we have now.

MIDDLEWARE

It generate the communication code between layer.
Blocking Interaction: "synchronous" command. Invoke → block → Execute → Response (Release)
Non-Blocking Interaction: invoking thread put the message in queue, the invoked queue when possible will read and will respond ("asynchronous")
Ofcourse blocking interactions are not suitable for distributed systems.
Remote Procedure Call: hides communication details behind a procedure call and help bridge heterogeneous application. When building a distributed application we don't need to know how it works internally. Other story if we build infrastructure (not our case)
How it works basically?



TRANSACTIONAL RPC

TRPC act as a bundle of RPC in which all of them are executed or none.
2PC = 2 phase commit: first gain consens from procedure calls to partecipate to the bundle then commit all of them.
In addition we can implement a 3PC to be more sure about the procedure calls (2PC anyway usually is enough)
Optimistical approach: most procedures commits → servant can execute in real-time when issued (rare case of abort, undo operation in that case)
Pessimistic approach: most procedures abort → servant executes after real commit (procedures buffered)
IMPLEMENTATION:
On client, developer can issue BOT (begin of transaction) and EOT (end ...).
Since we can have multiple clients, each transactions has an ID and context associated to them to keep track of operations.

MESSAGE ORIENTED MIDDLEWARE (MOM)

In this abstraction, client send messages to servant that are executed asynchronously (put them in queue)
Some "code" is used on client on in servant. In big systems we may have multiple servant on some shared queue. The basic idea is the one of publish/subscribe concept.
Subscription to messages can be done topic-based (receive only message with a specific topic) or content-based (filter on message content). Both method can be combined to be more precise.
Notification of messages can be done in push or pull mode depending on how messages arrive to destination.

WEB SERVICES

Remote Procedure Call through HTTP to access middleware methods
Service-Oriented Paradigm is the way to build programs using services as the constructs to support the rapid and low-cost development of distributed applications. Web service is a programmatically available application logic exposed over the internet i.e. any code/application deployed can be transformed into a network-available service. The idea is to make all modular. Services can be mixed together to create complete processes (eg. complete supply chain management) and they are platform independent.
Software-as-a-Service (SaaS) appeared first with Application Service Provider software model that "rents" applications to customer that subscribe them. Customer can only modify small things of User Interface.
The main cons were the inhability to develop interactive applications and to provide complete customizable ones.
ASP work typically in two way: ① User's machine access application with browser (only UI) or ② can download whole program to run it on his machine (SaaS)
Webservices are a tradeoff between these two approaches: only a small part of program runs on user's machine
They can be used within enterprise (enterprise application integration) or between enterprises (e-Business integration).
Informational Services: provide access to content by means of simple request/response
Complex Service: assembly and invoke many pre-existing services
Services are described by functional (behavior) and non-functional (stats) properties and can be stateful or stateless according to the fact that their context has to be preserved or not.
Coupling: indicates degree of dependency between two systems → Webservices are in fact loosely coupled
Granularity: the need or not of more interactions between other services: for example complex webservices are coarse-grained.

Synchronous services (RPC-style) deal with simple messages on request/response (eg weather request) while Asynchronous ones (document-style) typical interact with large files in asynchronous mode.

COMMUNICATION & COAP

Inter-application com: distributed technologies and object orientation. Weakness: server and clients must be implemented in some way (both JAVA for example) and hard to implement in presence of firewalls.

SOAP

Standard messaging protocol, encoding request and response using XML and sending theme over http.
Two possible communication styles: RPC (synchronous) and document (asynchronous)
1) An RPC web service appears as a remote object to the client. Client express request and wait response
2) Instead of a simple request, the body is an huge XML (can be non-structured too) and we may even not receive the response.
Both have Header: basic usage is to piggy-backing information at application level or any info that were not be possible to implement when SOAP was invented.
SOAP can both use GET and POST but only GET response and both POST request and response are soap messages.
(GET request = http normal request)
PROS: simplicity, portability, firewall-friendly, open, interoperability and universal accepted.
CONS: stateless, lot of HTTP and serialization by value and not by reference (XML request body that need to be handled)

SERVICE DESCRIPTION & WSDL

As in middleware, there must be a description for every service → WSDL
This is needed to let other service/app discover our service, with its XML interface.
Description of operations and structure.
Web Service Description Language is used to do this, an XML "contract" that describes the terms of connection to the service (What service does, Where it resides and How invoke it).
- Service interface definition: operations and their parameters.
- Service implementation: binds abstract interface to concrete network, protocol and data structure

WDSL Message Exchange

- One Way = no response after calling a procedure (void) → Notification if is the receiver to send the "request"
- Request/Response =  we have the response after RPC. → Sollecit/Response if " " "

REGISTRY & UDDI

Service Provider publish the service description in service registry. Service Requestor find it in the registry and can invoke it. → it works like a service store (STEAM for services)
Universal Description discovery and integration is a registry standard.
- white pages: key points of service
- yellow pages: division per taxonomy (argument)
- green pages: technical capabilities and info about services.

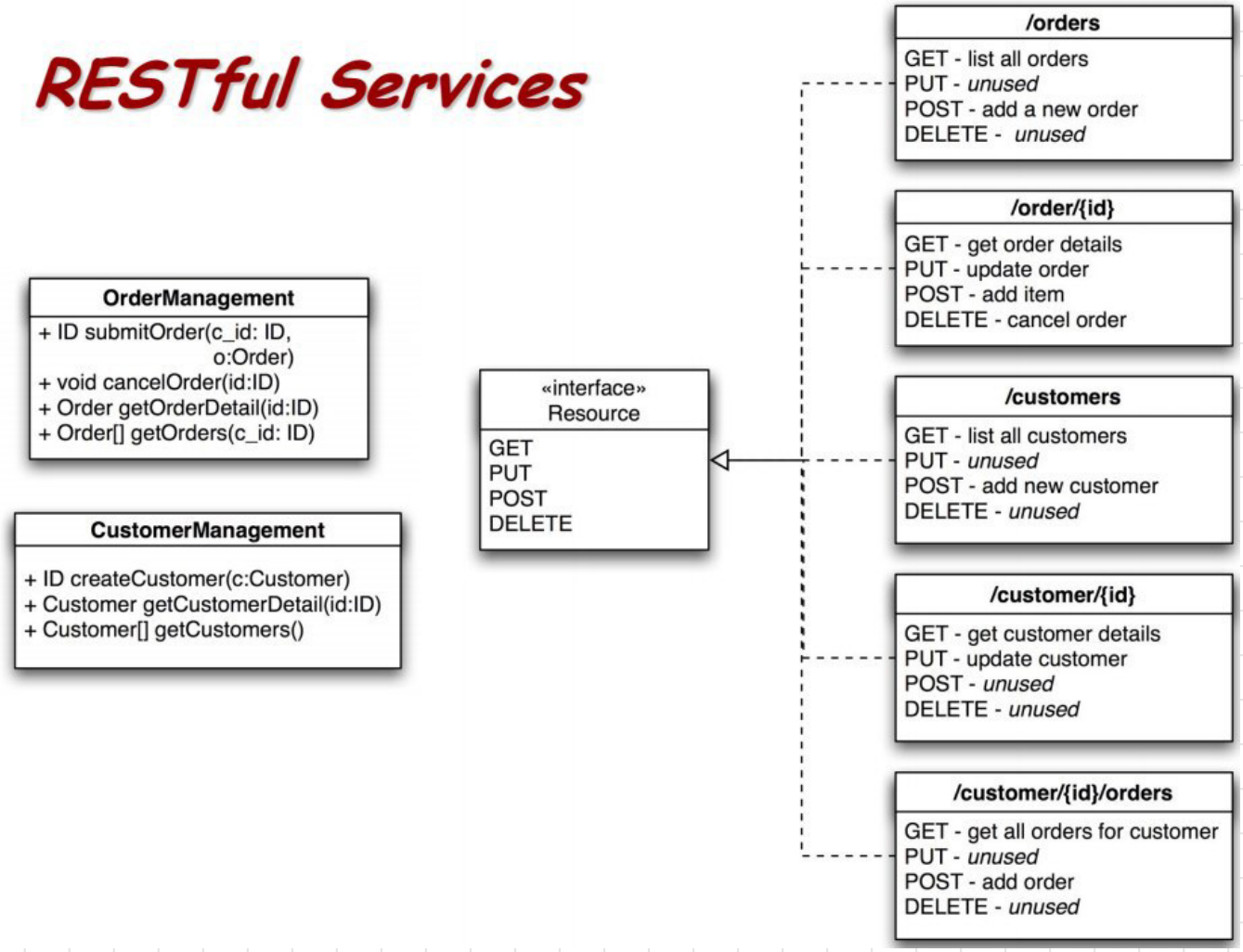
SERVICES MASH-UP

Web application that combines data from different sources in a single integrated tool. Lightweight form of composition based on various technologies: soap, RESTfull and Atom/RSS

The difference between WSDL-based and RESTful service is is that over messaging and single service layers there are 2 more layers on the first one: Multiple Interacting Services and Registry/Repository & Discovery.

RESTFUL SERVICES

Simple application interfaces trasmitting data over HTTP without additional layers as SOAP.
They make use of GET, POST (PUT) and DELETE on URIs.
In some way REST is a kind of RPC: getStockPrice = Current Stock Price.
Example:



Principles:

Addressability: each resource is on URL
Uniform Interface: HTTP REST
Stateless Interaction: no session
Self Describing Messages: metadata
Hypermedia: hyper links

SOFTWARE EVALUATION

Techniques to evaluate how much time/resources a software development will require.
Metrics can be direct or indirect.

DIRECT METRICS

L.O.C (Line of code): apparently is easy to compute (eg. cost per Loc) but it strictly depends on programming language and programming style. In addition we may have to compute also errors/KLOC (1K LOC) or documentation pages/KLOC

FUNCTION POINTS

Evaluate software at the border of it, between user and other application (software)
TL.DR. give points/weigth to different kind of operations, multiply for the times of calls and sum all together (more or less)

ILF: internal logical file, hold data through one or more elementary processes of the application being counted
EIF: external interface file, hold data of external application = the ILF of an other application.
EI: external input, process that manage ILFs to alter the behaviour of system
EO: external output, process that interact outside application (eg. present info to user), at least one mathematical function involved to create new data or alter the system behaviour
EQ: external inquiry, same as EO without mathematical functions or new data/alterations

Every function has a weigth: LOW, MEDIUM, HIGH.
In addition the value may be "adjusted" to capture general characteristics of the system (not in this course)
CONS: semantic difficulty, incomplete (internal functionality?), no automatism, different versions.
PROS: widely used, objective, usable in early development, technology indipendent.

Complexity of ILF and EIF depends on number of DET and RET
DET: data, element type, single field within a ILF/EIF
RET: record element type, group of field within a ILF/EIF
For operations instead
FTR: file type referenced, changed/read ILF or read EIF
+ DET

EFFORT ESTIMATION

Derive cost, effort and time from requirements.
Steps:
1) Requirements → Function points
2) FP → LOC
3) LOC → Time / Effort
4) Effort → cost

In addition we have to take into account the project parameters, external requirements for the project eg project platform.
We will focus on COCOMO, the process that derives effort from LOC (step 3).

COCOMO : Constructive Cost Model

Estimates effort M and optimal T and works on statistic of previous project using waterfall model
↳ not anymore

Basic formulas are $M = aS^b$ and $T = cM^d$ with S = effort in KLOC

Provides effort for :
- analysis and planning
- design
- development
- integration and test

M = Man Time : man time required to complete the Job (man-day, man-month etc)
T = Delivery Time : required optimal time to deliver software (years, months etc)
Manpower = M/T
Adjusting parameters estimate the context in which the software is developed

COCOMO 2.0

2 different models :
① Early design : little details for project initial phase
② Post-Architecture : more details for development and maintenance phase

BASICS of MEASURE THEORY

How far we are from final requirements.

5 scales :
- Nominal Scale : labelling variables without any quantitative value eg hair color → only equality test.
- Ordinal Scale : allow us to rank order of items we measure (eg student rank A, B, C, D) but not how much different between them (in some case adjacent ranks differ alot)
- Interval Scale : we can now compare the size of differences eg student's mark 28 and 26 → diff = 2.
  Need precise definition of scale unit!
- Ratio Scale : some of internal scale + absolute zero point eg Kelvin
- Absolute Scale : ratio scale with non negative values.

RATIO : result of division between two values of different and disjoint domains eg male/female. NOT Percentage
PROPORTION : division in which divident contribute to divisor eg $a/b+a$
PERCENTAGE : proportion normalized to 100
RATE : identifies a value associated with the dynamics of a phenomenon eg births per year/population
CRUDE if we calculate over all women (children and old ones too)

RELIABILITY of MEASURE

The consistency of our measurement = repeatability of measure
VALIDITY : the strength of our conclusions, inferences or propositions TLDR were we right?
Measure M = True Value T + Total Error E_T
$E_T = E_{systematic} + E_{random}$
↳ influences validity ↳ influences reliability

HYPOTHESES VERIFICATION

Compare different repeated measures. → perform statistics tests = challenging the hypothesis that the means of different samples are the same

null hypothesis: all true means are equals, observed differences are random

Test is performed fixing a priori the probability of error (α)

statistical hypothesis H: statement on distribution of one or more random variables

eg. Given two samples μ_a μ_b what is the probability that they come from same population?

- H_0 (null hypothesis) $\mu_a = \mu_b$ → same population
- H_1 (alternative) $\mu_a \neq \mu_b$ → different population

→ we define a formula on sample means capturing data differences generating random variable and p-value with respect to a threshold → p = probability of rejecting H_0 when true and we select for it a significance level α

So: $p > \alpha$ → accept H_0

$p \leq \alpha$ → accept H_1

FORMULAS

t-test: allow to compare the difference between the mean values of two samples, it exploits a comparison between means and standard deviation. Assuming the samples come from same population we can compute the probability density of t and the probability of t being equal or greater than a value

Given **two** samples of size n=7
a=(a1,...,a7)
b=(b1,...,b7)
from the **same** distribution
what is the probability p(t>=x)?

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

It depends on n=7
For ν=2*(7-1)=12
P(t>=1.356)= 20%
P(t>=1.782)=10%
P(t>=2.179)=5%
P(t>=2.681)=2%
P(t>=3.055)=1%
P(t>=3.929)=0.2%

0.025 · 2 = 5%

ν	0.10	0.05	0.025	0.01	0.005	0.001
1.	3.078	6.314	12.706	31.821	63.657	318.313
2.	1.886	2.920	4.303	6.965	9.925	22.327
3.	1.638	2.353	3.182	4.541	5.841	10.215
4.	1.533	2.132	2.776	3.747	4.604	7.173
5.	1.476	2.015	2.571	3.365	4.032	5.893
6.	1.440	1.943	2.447	3.143	3.707	5.208
7.	1.415	1.895	2.365	2.998	3.499	4.782
8.	1.397	1.860	2.306	2.896	3.355	4.499
9.	1.383	1.833	2.262	2.821	3.250	4.296
10.	1.372	1.812	2.228	2.764	3.169	4.143
11.	1.363	1.796	2.201	2.718	3.106	4.024
12.	1.356	1.782	2.179	2.681	3.055	3.929
13.	1.350	1.771	2.160	2.650	3.012	3.852
14.	1.345	1.761	2.145	2.624	2.977	3.787
15.	1.341	1.753	2.131	2.602	2.947	3.733
16.	1.337	1.746	2.120	2.583	2.921	3.686

Big t values are associated with little probabilities

If the actual t value is >= than t-crit value for 5% probability it is possible to accept H_1

86

σ^2 = varionce

σ = standard deviation

- H_0 true
  - reject H_0 → α type 1 error false negative
  - accep H_0 → $1-\alpha$
- H_0 false
  - reject H_0 → $1-\beta$
  - accept H_0 → β type 2 error false negative

ANOVA: analysis of variance, allow to compare two or more samples comparing the variability within (VAR_W) and between (VAR_B). Null hypothesis assumes all groups have same distribution. The idea is that if $W \gg B$ then the observed difference is caused by internal variability.

ANOVA on two samples = t-test

- $H_0: \mu_1 = \ldots = \mu_I$
- H_1: at least two are different

I samples of J items — Y_{ij} = j^{th} observation on i^{th} sample

mean of i: $\mu_i = \left(\sum_{j=1}^{J} Y_{ij}\right)/J$

general mean: $\mu = \left(\sum_{i=1}^{I} \mu_i\right)/J$

F-test

$$F = \frac{SS_B/(I-1)}{SS_W/[I(J-1)]} \qquad SS_B = J\sum_{i}^{I}(\mu_i - \mu)^2 \qquad SS_W = \sum_{i}^{I}\sum_{i}^{J}(Y_{ij} - \mu_i)^2$$

- accept H_0: $p > \alpha$ ($F < F\text{-crit}$)
- reject H_0: $p \geq \alpha$ ($F \geq F\text{-crit}$)

$$F = \frac{SS_B/(I-1)}{SS_W/[I(J-1)]}$$

I-1=2

I(J-1)=12

The probability that F>=3.89=0.05

numerator

nu \ nu	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	161	200	216	225	230	234	237	239	241	242	244	246	248	249	250	251	252	253	254
2	18,51	19,00	19,16	19,25	19,30	19,33	19,35	19,37	19,38	19,40	19,41	19,43	19,45	19,45	19,46	19,47	19,48	19,49	19,50
3	10,13	9,55	9,28	9,12	9,01	8,94	8,89	8,85	8,81	8,79	8,74	8,70	8,66	8,64	8,62	8,59	8,57	8,55	8,53
4	7,71	6,94	6,59	6,39	6,26	6,16	6,09	6,04	6,00	5,96	5,91	5,86	5,80	5,77	5,75	5,72	5,69	5,66	5,63
5	6,61	5,79	5,41	5,19	5,05	4,95	4,88	4,82	4,77	4,74	4,68	4,62	4,56	4,53	4,50	4,46	4,43	4,40	4,37
6	5,99	5,14	4,76	4,53	4,39	4,28	4,21	4,15	4,10	4,06	4,00	3,94	3,87	3,84	3,81	3,77	3,74	3,70	3,67
7	5,59	4,74	4,35	4,12	3,97	3,87	3,79	3,73	3,68	3,64	3,57	3,51	3,44	3,41	3,38	3,34	3,30	3,27	3,23
8	5,32	4,46	4,07	3,84	3,69	3,58	3,50	3,44	3,39	3,35	3,28	3,22	3,15	3,12	3,08	3,04	3,01	2,97	2,93
9	5,12	4,26	3,86	3,63	3,48	3,37	3,29	3,23	3,18	3,14	3,07	3,01	2,94	2,90	2,86	2,83	2,79	2,75	2,71
10	4,96	4,10	3,71	3,48	3,33	3,22	3,14	3,07	3,02	2,98	2,91	2,85	2,77	2,74	2,70	2,66	2,62	2,58	2,54
11	4,84	3,98	3,59	3,36	3,20	3,09	3,01	2,95	2,90	2,85	2,79	2,72	2,65	2,61	2,57	2,53	2,49	2,45	2,40
12	4,75	3,89	3,49	3,26	3,11	3,00	2,91	2,85	2,80	2,75	2,69	2,62	2,54	2,51	2,47	2,43	2,38	2,34	2,30
13	4,67	3,81	3,41	3,18	3,03	2,92	2,83	2,77	2,71	2,67	2,60	2,53	2,46	2,42	2,38	2,34	2,30	2,25	2,21
14	4,60	3,74	3,34	3,11	2,96	2,85	2,76	2,70	2,65	2,60	2,53	2,46	2,39	2,35	2,31	2,27	2,22	2,18	2,13
15	4,54	3,68	3,29	3,06	2,90	2,79	2,71	2,64	2,59	2,54	2,48	2,40	2,33	2,29	2,25	2,20	2,16	2,11	2,07
16	4,49	3,63	3,24	3,01	2,85	2,74	2,66	2,59	2,54	2,49	2,42	2,35	2,28	2,24	2,19	2,15	2,11	2,06	2,01
17	4,45	3,59	3,20	2,96	2,81	2,70	2,61	2,55	2,49	2,45	2,38	2,31	2,23	2,19	2,15	2,10	2,06	2,01	1,96
18	4,41	3,55	3,16	2,93	2,77	2,66	2,58	2,51	2,46	2,41	2,34	2,27	2,19	2,15	2,11	2,06	2,02	1,97	1,92
19	4,38	3,52	3,13	2,90	2,74	2,63	2,54	2,48	2,42	2,38	2,31	2,23	2,16	2,11	2,07	2,03	1,98	1,93	1,88
20	4,35	3,49	3,10	2,87	2,71	2,60	2,51	2,45	2,39	2,35	2,28	2,20	2,12	2,08	2,04	1,99	1,95	1,90	1,84
21	4,32	3,47	3,07	2,84	2,68	2,57	2,49	2,42	2,37	2,32	2,25	2,18	2,10	2,05	2,01	1,96	1,92	1,87	1,81
22	4,30	3,44	3,05	2,82	2,66	2,55	2,46	2,40	2,34	2,30	2,23	2,15	2,07	2,03	1,98	1,94	1,89	1,84	1,78

denominator

F-critical values at α = **0.05**

I=2

J=7

$$\mu_a = (1+6+1+1+6+6+2)/7 = 3{,}28$$

$$\mu_b = (5+3+1+6+2+4+1)/7 = 3{,}14$$

$$\mu = (1+6+1+1+6+6+2+5+3+1+5+2+4+2)/14 = 3{,}21$$

$$SS_B = 7[(3{,}28-3{,}21)^2 + (3{,}14-3{,}21)^2] = 0{,}0714$$

$$SS_W = (1-3{,}28)^2 + (6-3{,}28)^2 + (1-3{,}28)^2 + (1-3{,}28)^2 + (6-3{,}28)^2 + (6-3{,}28)^2 + (2-3{,}28)^2 + (5-3{,}14)^2 + (3-3{,}14)^2 + (1-3{,}14)^2 + (6-3{,}14)^2 + (2-3{,}14)^2 + (4-3{,}14)^2 + (1-3{,}14)^2 = 62{,}29$$

$$F = \frac{0{,}0714/(2-1)}{62{,}29/[2(7-1)]} = 0{,}01376 \quad << \quad F\text{-crit}_{1,12} = 4.75$$

→ **Accept H_0**

MICROSERVICES

Monolithic softwares are hard to maintain and evolve, need lot of time to build/test/release updates and adding features is complicated
Microservices instead are separated modules that can interact with each other and this is the current way of software development. Each microservice can be updated on its own. Every microservice should be devoted to a single functionality.

Anatomy: data store, application logic and public API

Principle 1: microservices only rely on each other's public API
Principle 2: use the right technology for the job (eg. different frameworks talking together)
Principle 3: secure the service (gateway, authentication, secret management)
Principle 4: ask for permission to access third's party services
Principle 5: Conway's law, software architecture is a copy of organization's communication structure
Principle 6: automate everything is possible to automate (eg. test)

DEVOPS

Software development + Operations in loop.
Main reason of this new implementation is to fix problems like slow releases or bug fixes.
- Increase velocity (btw produce - release)
- Reduce Downtime
- Reduce Human Error (test in advance)

Anyway DevOps is not 100% automated and there isn't only one way to do them!
1. Eg. Team Development is in charge of change/modify/test features while Team IT is in charge of enhance stability and services → quality is a responsability for both! → collaboration
2. Bridge collaboration with automation. Enable continous delivery to end users unifying development with operations. (+ quality assurance)
3. Developer → code repository → build automatically → test → deploy → monitor and improve → repeat.

With DevOps we "unify" development and operations decreasing issues between the two category.

CONTINOUS INTEGRATION: use a mainline code base (repository), no developers isolation.
Frequent commits, automation in building and testing.
PRO: all updated, early bug detection, fast release
CONS: lot of work to automate test and building process, partial commits can generate issues.
CONTINOUS DELIVERY/DEPLOYMENT: automation of delivery/deployment. The only difference is that the first one has a manual part (deployment to production from acceptance tests). Delivery ⊂ Deployment
After every internal operation, developers gets feedback eg after acceptance test

TOOLS: they are divided in collaboration tools (eg Slack), planning tools (eg. Asana), Source control tools (eg. Git), Issue tracking tools (eg. Jira), configuration management tools (eg Puppet), Database DevOps tools (eg DBMaestro), Continous integration tools (eg Jenkins), Automated testing tools (eg Telerik) and Deployment Tools (eg IBM uDeploy)