SOFTWARE ESTIMATION

Imperative to start knowing how to estimate the cost of software development.

direct metrics

- line of codes & more practical
- how many decision points are there

NOT VERY OBJECTIVE, since more line of code != more content

requirement metrics

- how many functionality does the sw offer?

FUNCTIONALITY is what usually should get used, more objective.

indirect metrics

- decide how much part is this sw of of a bigger service (e.g. transactions in Netflix service

DIMENSION OF METRIC

- Line Of Code (LOC)

usually it's used as 1000 LOC (kLOC)

RATES:

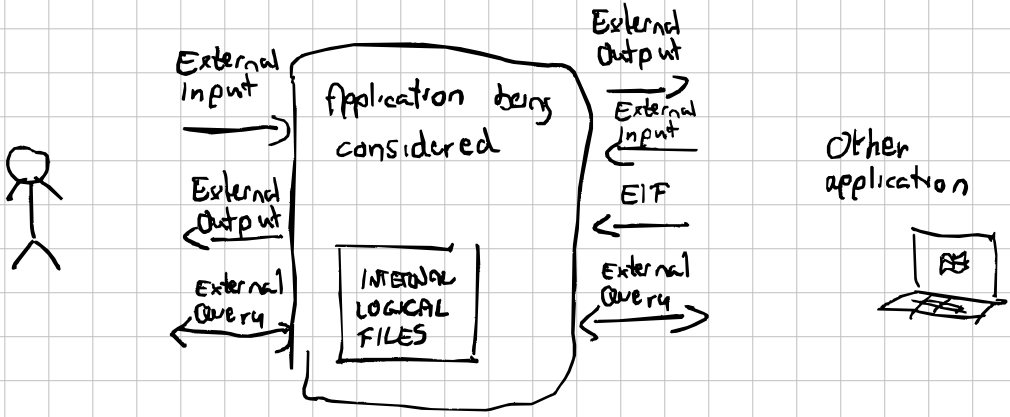
- internal error / kLOC

it's dependant on the programming language and style of code.
ISO standard DOESN'T use LOC to estimate software.

Function Point

Proposed in '79, it's an empirical formula based on Functionality (weighted).
ISO standardizes and expands the proposal.

Functionalities



External Output is more costly than EQ

The proposal

	Count	weight	
EI	x	3-4-6	=
EO	x	4-5-7	=
EQ	x	3-4-6	=
ILF	x	7-10-15	=
EIF	y	5-7-10	=

Internal Logical File: hold data of control information that has relevance with boundary of the application. It's USER-IDENTIFIABLE (Databases?)

External Interface File: Like ILF, user-identified, but maintained by an another application. Hold Referenced data through one or more processes. It's not maintained primarily by our application

TRANSACTIONS

External Input processes data that comes from OUTSIDE the application boundary.

External Output: Presents information to the user to processing logic and through retrieval of content. The processing logic must contain at least ONE MATHEMATICAL FORMULA (to make more fair)

External Inquiry: sends data outside the application boundary. Used to present information to present user through retrieval of ILF or EIF. Shouldn't have any math.

How to compute complexity

DET Data Element Type: user-identified single field within ILF/EIF (simple datatype)

RET Record Element Type: user-identified group of fields within ILF/EIF

ILF / EIF complexity varies, Depending on needed DET/RET

Ret	1-19 Det	20-50 Det	51+ Det
1	Low	Low	Med
2-5	Low	Med	hig
6+	med	hig	hig

FTR (File Type Referenced), read/wrote ILF OR READ EIF

FUNCTION POINTERS

It's a way to objectively determine the functions in the application through analysis of 14 indicators, and from there get LOC ESTIMATION

Each indicator gets assigned a value between:

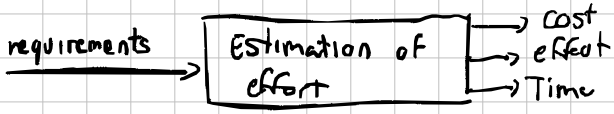
0 (not relevant) - 5 (essential)

$$AFT = Total \times \left(0.65 + \underbrace{0.01 \sum_{i=1}^{14} F_i}_{\text{corrective factors.}}\right)$$

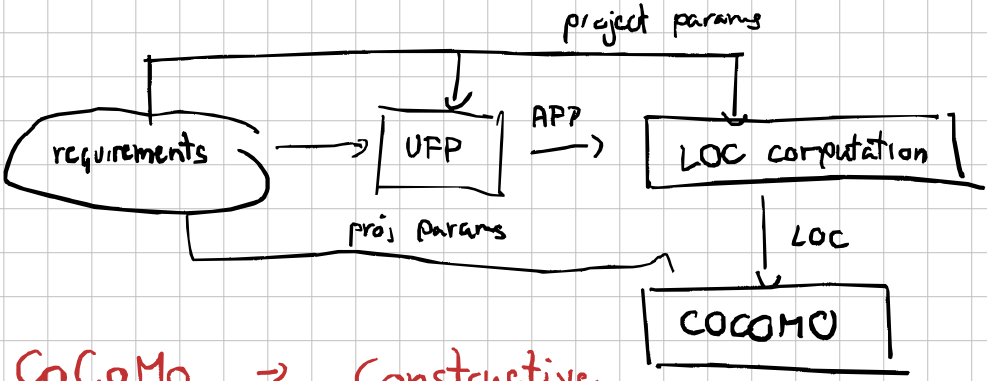
- It's a programming-independent way to assess functionality
- accurate
- calculation is objective
- certifications for it

- Legacy terminology
- incompleteness
- different revisions

EFFORT ESTIMATION



- From requirements to function points
- FP → LOC
- LOC → Time/Effort
- Effort → COST



CoCoMo → Constructive Cost Modelling

Estimation derived from FP analysis are usually pretty optimistic, needs to be done some corrective measurement on sw houses part

↳ This happens the more the project is bigger

COCOMO

Estimates the effort M and optimal time T
Methods works by STATISTIC, and with idea waterfall models.
This considers perfect requirements, which they usually aren't.

It provides an effort indication based on four phases:
- analysis and planning
- design
- development
- integration and test.

M: Effort
Man time to develop the project

T delivery time
required optimal time to deliver working software

Manpower
effort across time: represents number of people working during project execution.
Ideally, manpower = EFFORT / DELIVERY TIME
THIS DOESN'T WORK, needs interaction cost between

we need adjusting parameters: we have several parameters evaluated over 6 value, from very low to extra high

We have 3 model types, based on the detail level we want to have based on if the project is organic, semi-detached or embedded.

In COCOMO requirement analysis and planning are not considered, since they shouldn't be done in sw side!

Usually we usually have man-months

We can consider an organic model

S = 32kLOC

M = $2.4(32)^{1.05}$ = 91 MM

T = $2.5(91)^{0.38}$ = 14 months

People 91/14 = 6.5

productivity 32k/91 0.382 kLoc/months

18.5 Loc/day ???

This seems low, but it's because in cocomo we consider that a software programmer doesn't just "write code"

This is original cocomo

$$M = aS^b$$

$$T = cM^d$$

a, b, c, d — parameters derived from project type
- organic
- semi detached
- embedded

effort is calculated in person-month

Cocomo has error rate of 20%/65% of estimations, way too optimistic

COCOMO II

Motivations
- new lifecycle sw remodels
- Reusal of code
- different levels of estimation precision [early-design / post-architecture]

Based on interpolation of statistics. Range and uncertainty of estimation precision lessens as next phases we have.

deals with the different type of design

$$PM = A \times SIZE^{E} \times \prod_{i=1}^{n(7,17)} EM_i$$

SIZE in KLOCs

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j^{(1)}$$ (some of cost drivers)

lower is higher

SF scaling factor

E: economy - diseconomy scales

$$TDEV = C \times (PM_{adj})^{F} \times SCED\%/100$$

TDEV develop. time

$$F = D + 0.2 \times 0.01 \times \sum_{j=1}^{5} SF_j$$

EM: effort multiplier

PM: person-months

in the current version of actual calibration:

A = 2.94
B = 0.91
C = 3.67
D = 0.28

this is what works now, in the future these may change!

(1) scaling factors can be found on cocomo.26

FLEX (flexibility) PREC (precedentness) RESL (risk resolution) TEAM (+ear cohesion) PMAT (process maturity) based on CMM

It's more in depth than original Cocomo, giving time estimation on different subsystems

Table in page 33, SE_Cocomo

BACKFIRING

Based on statistics, tells us how much LOC per function pointer we need:

Ex

invoice db

- 51 Fp
- C language
- C backfiring 128
- LOC is 51 * 128 = 6.5 KLOC

NOMINAL PARAMETERS used.

EFFORT MULTIPLIERS

Product

RELY	Required sw reliability	Indipendent of team
DATA	Database size	
CPLX	Product complexity	
RUSE	Reusability	
DOCU	documentation level needed	

System

TIME	Time constraint	Indipendent of team
STOR	storage constraint	
PVOL	Platform volatility	

Personal

ACAP	Analyst capability	choosing appropriate people
PCAP	Programer capability	
APEX	Application Experience	
PLEX	Platform Experience	
LTEX	Language & Tool experience	
PCON	Personnel continuity	

Project

SITE	Multisite dev.	
TOOL	use of sw tools	
SCED	schedule constraints	

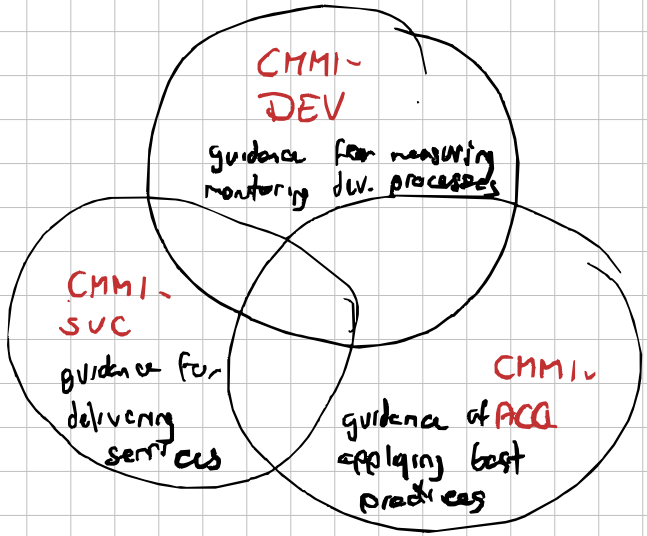
Process Model

It's a structured collection of practices that describe characteristic of effective processes, proven by experience,
It's used to:

- set measurable objectives and priorities
- ensure stable, capable and mature processes
- improve projects
- diagnose/ CERTIFY state of organization

CMMI is the answer to request of a model integration from DoD. It provides organizations with elements for effective processes,

CMMI Framework is structure that organizes components used in generating models, and every component is organized in constellations.
3



CMMI (Capability Maturity Model Integration)

Levels of achievements of a company defined in a pyramid:

LEVEL 0: Incomplete, no generic goals exist for this level
LEVEL 1: Performed, supports and enables the work needed to produce work products. Everything happens ad-hoc
LEVEL 2: managed, process that has basic infrastructure in place to support the process. Planned and executed with a policy. Has organizational visions, even in periods of stress
LEVEL 3: Defined, very well structured, tailored from the organization's set of standard processes.
LEVEL 4: Quantitatively managed; process that are defined are controlled via statistics and quantitative measurements, its managed throughout the life of the process.
LEVEL 5: Optimizing: improving process based on understanding of the common causes of variations inherent on the process itself. The measurements of the past improve the futures

PROCESS AREAS

General goals and practices apply to multiple process areas
- Performed
- managed
- defined

each PA has 1 to 4 goals, each goal is comprised of specific practices

CMMI-DEV: 22 process areas (page 13)

PROCESS MGMT | PROJECT MGMT | ENGINEERING | SUPPORT

the more you want to go up the pyramid the more PAs you need

ISO 12207 Standards for software lifecycle processes

Standard that "deals with sw engineering"

Defines structures and activities involved in the sw dev process.
FUNCTIONAL APPROACH: activities that go from input to output

- Five lifecycle processes related to primary involved agents:
  buyers, suppliers, developers, maintainers, operators, managers, and technicians
- Eight supporting life cycle processes
- Four organizational processes.

The standard is based on two basic principle
MODULARITY RESPONSIBILITY

5 Primary life cycle processes

5.1 Acquisition
5.2 Supply
5.3 Development
5.4 Operation
5.5 Maintenance

6 support processes

6.1 Documentations
6.2 Config. mgmt
6.3 Quality assurance
6.4 Verification
6.5 Validation
6.6 Joint Review
6.7 Audit
6.8 Problem solving

7 Organizational processes

7.1 Management.
7.2 Infrastructures.
7.3 Improvements.
7.4 Training.

Information system is an organized system for the organization, collection, communication and storage of information.
↳ can, but it's NOT NECESSARILY NEED TO, be software, or also papers.

Core Process Activities

- **Specification:**
  in this part we establish what our service/s needs to do:
  - Functional Requirements
  - Non-Functional Requirements (quality)

  This is done through a 4 step process:
  1. Feasibility study
  2. Requirements elicitation
  3. Requirements specification
  4. Requirements validation (are this requirements what I really asked)

  In this part we create a system model, the user's and the system requirements, and we create a document

- **Design and Implementation**
  It's the process around going from specification to an actual running system. This goes through 2 different sub-phases:
  - Software design, that goes through a structured design of the software without programming it
  - Implementation, in which the actual LOC are written and the system goes to a running state

- **Validation**
  Unit & System testing, debugging phase

- **Evolution**
  sofwar needs to be thoroughly documented, easily maintainable

SCRUM

- Agile process that focuses to deliver the most possible for a short timeframe
- allows to rapidly and repeatedly inspect software (every 2 weeks, 1 month)
- Business sets priorities, teams self-organize to deliver the highest priority features
- every two weeks see release and choose if continue to enhance.

used by lots!

Characteristics:

-> self organizing, in peer-to-peer way
-> system processes in 2-4 weeks sprints!
-> requirements are captured as items in "product backlog"
-> No specific engineering practices required
-> one of the agile process

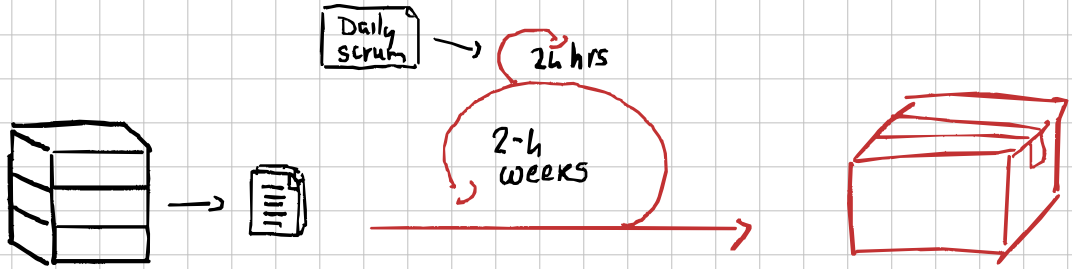
AGILE MANIFESTO

Individuals and Interaction	over	process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation

Self organizing teams

Scrum Sprints

We se with sprints of 2/4 weeks, iterative process



product backlog — Sprint backlog — sprint — Shippable product increment

Sequential vs overlapping development

spiral development is similar to scrum! - but with way less documentation -

we shouldn't do changes on the sprint!

if we fail the sprint we can repeat the sprint again.

SCRUM FRAMEWORK

Roles

- product owner (EXECUTIVE) — responsible for ROI, defines features, accepts or rejects work results
- scrum master (ORGANIZER) — represents management of process, impediments, team cooperation assurance
- team — : 5/9 people, cross-functional, and full-timers, self organized and interleaved. Shouldn't be changed during sprint!

[Waterfall processes have one single role per person]

Ceremonies

- Sprint planning meeting.
  After a planning in which we define team capacity, product, tecnology we produce a spring goal, analyzing the product backlog. There will be sprint planning, creating the backlog.
  - item selection from backlog, assigning task and putting ETAs on work
  - high level design considered
- Daily Scrum
  Daily meeting that the team should do
  - standing up
  - 15-minute meetings

  It's not for problem solving, but it's to assign day-to-day tasks
  HELPS to avoid unnecessary evenings.
  Usually based on 3 questions approach:
  what did you do yesterday?
  what will you do today?
  is anything in your way?
- SPRINT REVIEW
  The team presents product of the sprint, usually through a demo of new features
- SPRINT RETROSPECTIVE
  Periodically look at issues of team, product etc...
  It's done after every spring along review.

Artifacts

- **Product Backlog**
  It's the actual requirements collected as a list of what the software should do, which is ordered by the owners' priority
- **Sprint Backlog**
  part of product backlog, chosen to be the objective of the sprint
- **Burndown chart**
  chart with analysis of what has been done, (non-increasing)

USER STORIES

Agile development processes create requirements as "user stories", which usually take the forms of

"As a [user role], I want to [action], so I can [reason]"

with every user story there will be a story point associated with it, rating the effort needed to implement it (usually scale 1-10)

Every sprint will have a goal, which is a short statement of what the work will be focused during the sprint

PREREQUISITES COLLECTION

User stories should be:

- SPECIFIC } implies known good input and expected result exist
- MEASURABLE } and expected result exist
- ACHIEVABLE - complete in 1 iterations
- RELEVANT - that has a business value
- TIMEBOXED - if we finish time STOP STORY, avoid underestimating length
  - in case reschedule

[GIVEN condition WHEN : do action THEN things should happen]

[answer "why" 5 times !!!]

Building a successful UI

- SaaS often face users, we need a UI
  - ↳ we need to interact with customers to create UI design, so that its complete?
    - Avoid What-I-Said-But-not-what-I-want !

we need to have a Lo-Fi UI with pen and papers, literally drawing it!
- ↳ very easy to draw and to change

[Mecalli wants Lofi mockup]

create a storyboard !!!

SOFTWARE QUALITY

(AND ISO 25010)

12207 Software development processes (ichi alternatives)

SW PRODUCT set of { programs, procedures, documentation, data associated }

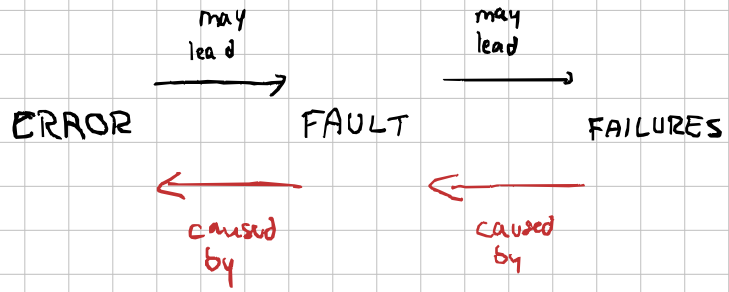
ISO 25010 specifically evaluates quality of SOFTWARE

4 COMPONENTS

1. Code
2. Procedures: all needed information for proper SW execution (sequence, people, roles, policies...)
3. Documentation handling for:
   - Development
   - User
   - Maintainers
4. Associated data, files, DBMS, test reports ...

TYPES

- ERROR: committed by human being in SW dev. process
- FAULT (or defect) is physical characteristic of SW producted by an error
- FAILURE is result of defect that can occur during the use of the software product and is PERCEIVED BY USE



HOW TO AVOID ERRORS

we usually don't need to have 0 errors, but we want to avoid faults / failures. we can do this by analyzing all error sources.

SOURCES:

1. Defective requirements:
   - Wrong
   - Missing
   - Incomplete
   - Useless
2. Customer - Developer misunderstanding:
   - official requirements
   - official / verbal changes.
   - query - answer.
3. Deliberate deviations from requirements.
   - reusement of sw
   - official / unofficial omission (time/money motivation)
4. Design errors
   - algorithms
   - misunderstanding
   - errors on boundary conditions (input fuzzing) } can lead to exploit
   - internal state omissions
   - missing defined behaviour for (error) input
5. Coding Error
   [...] big parenthesis on error types!
6. Non - Standard Coding
7. Test phases too short
   - missing / incomplete test plans
8. Documentation error:
   User manual is wrong, design and/or software documentation is wrong

Data Reference / Data Declaration Error

- referring of not initialized value
- badly named / duplicate values!

Computational errors

- Inconsistent types in computation?
- different precision error
- overflow & underflow
- division by zero
- Expressions correctly formatted

Comparison error

- incompatible comparison
- right comparison operator
- boolean expression correct?

Control flow error

problems inherent with the algorithm itself, usually

Interface errors

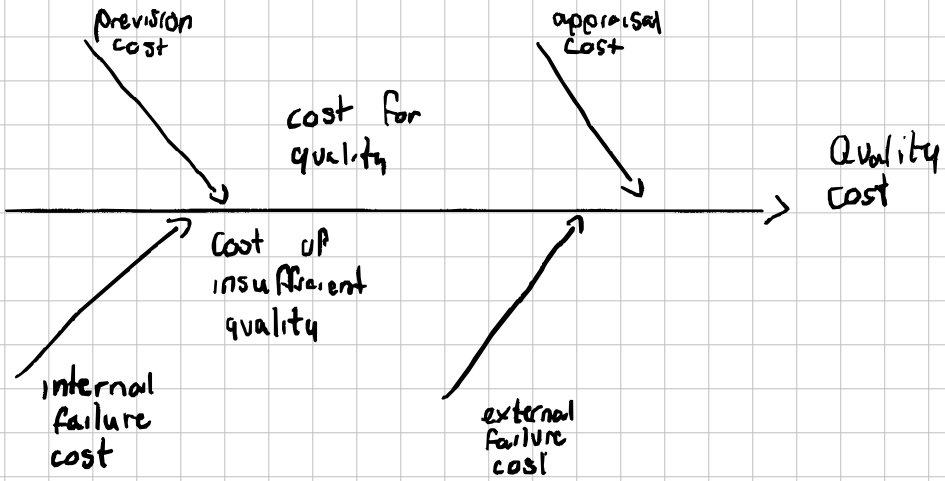
- correct order/number of parameters?
- data type correct? [sanitization!]
- value of reference?
- correct measurements units?

IO ERROR

- Right file types?
- EOF handled?

QUALITY

costs:



what do we want to pay for quality?

Usually people pay less than possible

- GOOD
- FAST
- CHEAP

we can usually choose two

what is sw quality?

IEEE says: "level at which a system, component or process meets the requirements, and also level at which they MEET needs and expectations of a user.

SOFTWARE QUALITY ASSURANCE

Set of actions that allow to have confidence that product conforms to set of technical requirements.

Evaluate the process of sw productions.

To have good sw...

... we want to have good quality requirements.
In particular, we need quality requirements.

- USER REQUIREMENTS
- TECHNICAL SPECIFICATION

Quality factors

In 1991 ISO 9127 we have Software engineering - Product quality, revised and published in 2001 built up on McCall and Boem models.

In 2011 was revised as 25010. Called SQuaRE system and software Quality Requirements and Evaluation

it's unlikely that all of SQuaRes point are applicated independently in perfect way.
It's also very abstract, it's not a guide on "how" to write good software.

Objectives

recipients are:

- users
- developers
- Client
- Sys admin
- Customers

QUALITY IN USE

- Effectiveness : # of reached objectives
- Efficiency
- Satisfaction : usefulness, trust, pleasure, comfort
- Freedom from risks : Economics, Health, Environmental risk mitigation (safety incident rate)
- Context coverage : Completeness, Flexibility, # of successful specification coverage

PRODUCT QUALITY

see later ↳

Product Quality Model

Functional suitability
- completeness
- correctness
- Appropriateness

Efficiency
- time complexity
- resource utilization
- Capacity

Compatibility
- co-existence
- interoperability

Usability
- Appropriateness Recognizability
- Learnability
- operability
- user-error protection
- Aesthetics of interface
- Accessibility

reliability
- Maturity
- Availability -> uptime
- Fault tolerance
- Recoverability

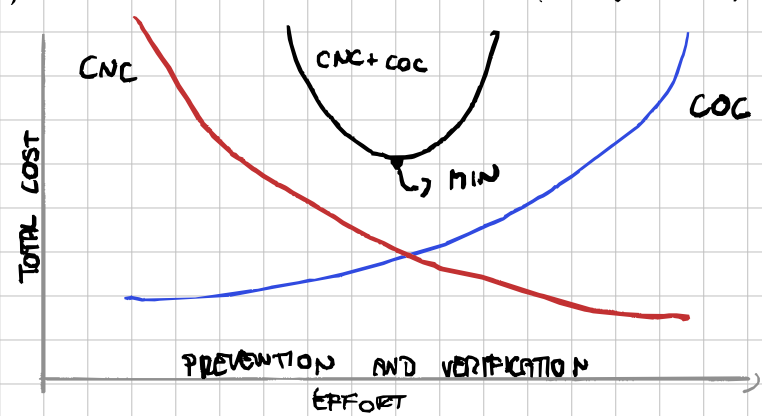
Security
- Confidentiality
- Integrity
- Non-repudiation
- Authenticity
- Accountability

Maintainability
- documentation - writing
- Modularity
- Reusability
- Analysability
- Testability

Portability
- adaptability
- replaceability

QUALITY COST

It's an equilibrium between costs of non compliance (lack of quality) and conformity (cost of applying quality)



Qualities are SOFTWARE SPECIFIC!

ORGANIZATIONS

	General	Electrotechnical
International	ISO	IEC
European	CEN	CENELEC
National	UNI	CEI

International Standard Organization

International Electrotechnical Commission

Comitato Europeo di Normazione

Comitato Europeo di Normazione Elettrotecnica

Ente Nazionale di Unificazione

Comitato Elettrotecnico Italiano

(ANSI - American National Standard Institute)

MEASUREMENTS

3 Types of measurements

- DIRECT: Detectable without influence of external factors, eg. reading documentation & source code
- NON-DIRECT: Derived from measurements of one or more attributes -eg. run-time, response time-
- INDICATORS: measurements derived indirectly from other measurements, inferences,

Every measurements has:

- Dimension (absolute / ratio scale) → Memory footprint
- Occurrences (nominal / absolute scale) → number of comp. warns
- Time (absolute / ratio scale) → Time to elaborate data

How do we create measurements?

dividing by time
- rates / frequencies / Time proportions
  occ time Dimension

occurrences
- occurrency prop / mean / Time interval
  occurrence Dimension time

Size:
- Densities / Efficiency / Dimension Proportion
  count time size

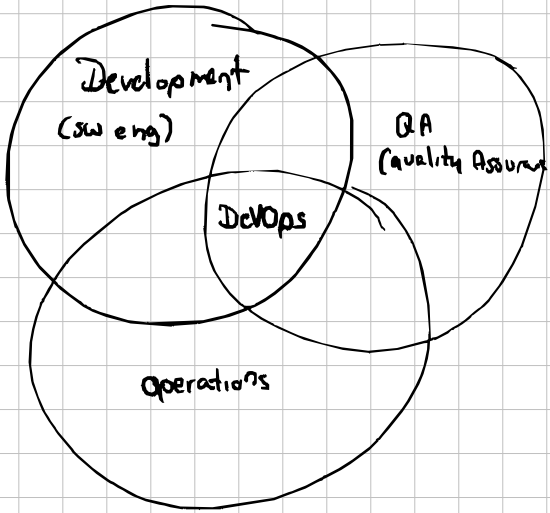
In mixing measurements we can create indicators of usability, efficiency, availability, ...

DevOps

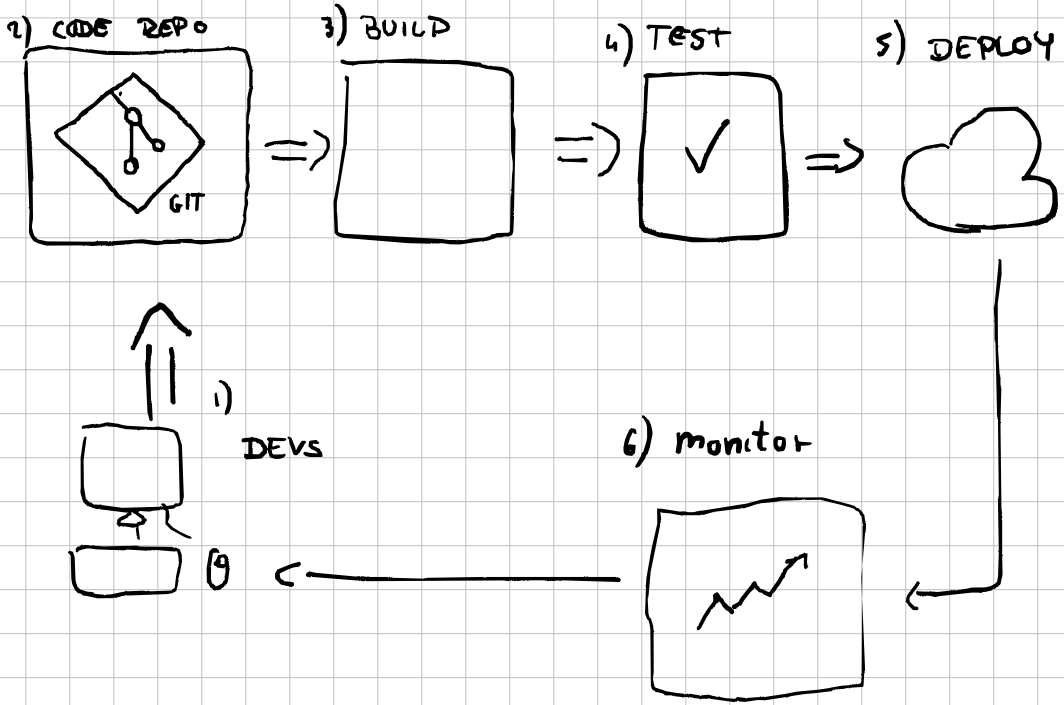
means synergy of sw development and operations.

why ?
It's a tool for automating some of the processes. It increases velocity, reducing downtime and human errors.
It's proven by experience to be working.

DevOps creates a sense of shared responsibility, increasing collaboration across development and "business", and AUTOMATES the software delivery process and infrastructure changes.



The basics



Continuous cycle

We are improving AGILE development that is an iterative approach that emphasizes importance of quick software delivery Collaboration between customer and developer for continuous improvement

CONCEPTS

CONTINUOUS INTEGRATION (CI)

It's the process of integrating code into a mainline code base.

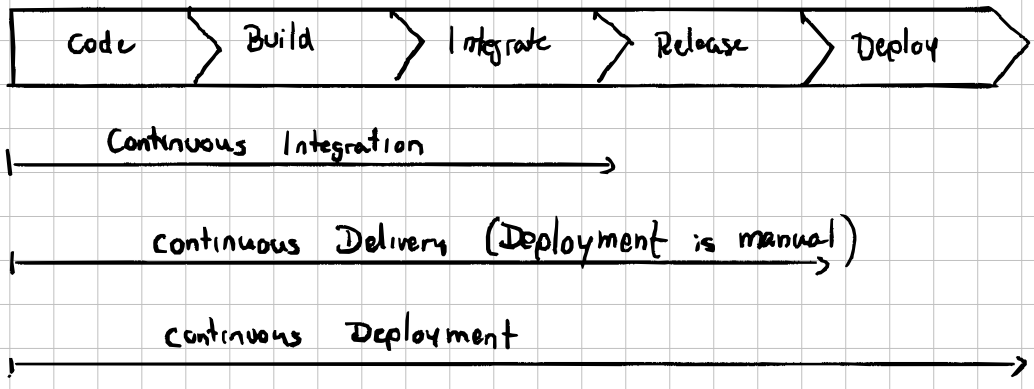
Uses the concept of a SOURCE CONTROL SERVER, that integrates developers' code into shared repository multiple times a day.

May be difficult to set up first, especially for the automated testing phase. PARTIAL code could completely make any test fail.

CONTINUOUS DELIVERY/DEPLOYMENT (CD)

having continuous delivery means that software is always production-ready, being able to be released quickly and automatically.

continuous deployment means to actually release the good builds to the users, it's next step to continuous delivery



Creating DevOps toolset

- **Collaboration**: helps the collaborations, sharing knowledge by communication (eg. Slack, Teams)
- **Planning**: provides transparency to stakeholders, showing what planning is (Jira)
- **Source Control**: Tools that make up building blocks for entire process, like code, documentation, sources etc. (e.g. Git)
- **Issue Tracking**: Increase visibility of issues. Should be unified. (e.g. Jira)
- **Configuration management** Tools to guarantee consistency at scale by using one tool for configuration. (e.g. Puppet, Chef)
- **Database devops** Database tool. What else should be said? (eg. DBmaestro)
- **Continuous Integration**: Provide feedback loop by merging code and doing tests (e.g. Jenkins)
- **Automated testing**: They test the code. Needs to be quick for better integration (see Telerik)
- **Deployment**: Tools to do the actual deployment. Does continuous delivery

SysML

Creations of systems, not just softwares! Abstract modeling of generalization of complex system
- Bike/car sharing.

different parts:

BDD - Class diagrams with we want to design system, (Very similar to UML)
Block Definition Diagram

Associations possible

- Generic: Two blocks "cooperate" in some way
- Composition: Component blocks can exist in context of the owner "composite" block
- Aggregation: Composite contains the components, but components exist outside the composite
- Generalization/specialization: specialized blocks has all the properties of generalized, but can add some more

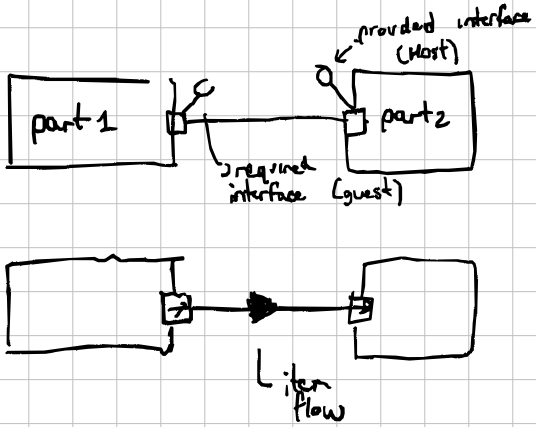
INTERNAL BLOCK DIAGRAM:
- defines the use of blocks in composition.
Internal structure becomes explicit, along with signalling and communication

SysML Ports

- Specifies interaction point on blocks and parts
- Integrate behaviour with structure
- Syntax portName : TypeName

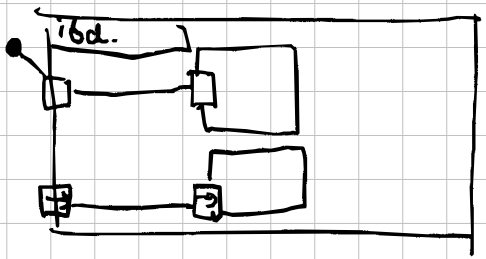
Types of ports:

- Standard Port (UML) operation oriented
- Flow Port - uses for physical signals, and are typed by blocks



delegation used to present encapsulation!

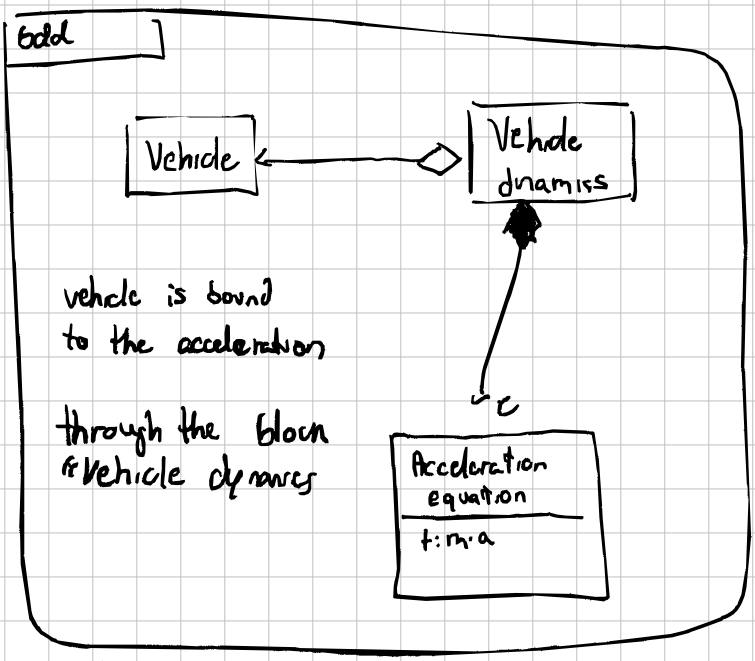
connection can cross boundary without nested hierarchy.



PARAMETRIC DIAGRAMS

Can be used to express constraints between value properties.
Constraint are defined by another block

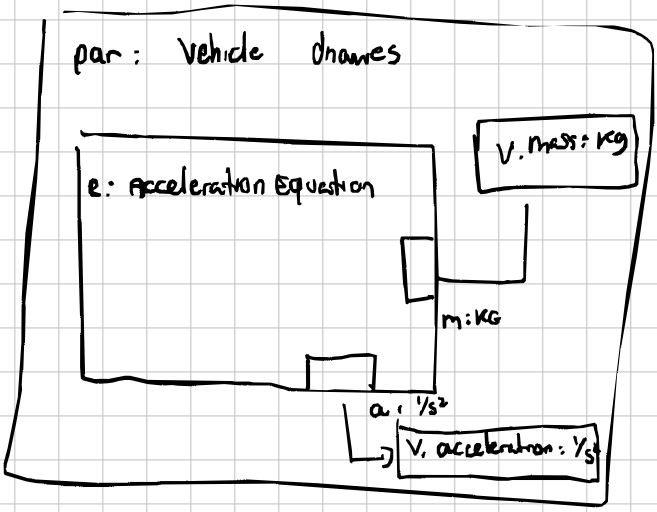
Binding of constraints parameters to value properties of blocks.
Ex. vehicle mass bound to param m in F: m×a)



pg 48. SysML

They are STATIC DIAGRAM! -out of scope of SysML-

useful for performance properties, computational engine is OUT OF SCOPE from SysML



pg 50. SysML

OCL is strong candidate for specification in this

BEHAVIOURAL DIAGRAM

Activity diagram

transformation of input in output through controlled sequence of actions.
(similar to dataflow)

SysML adds support for continuous flow modelling (continuous-time system)

Represent wide range of applications:
- Tradeoff between generality vs understandability

Application spectrum is between two endpoints according to way activities accept input and provide output.

Non-streaming activities: activities that accept input only when start, and output only at the end.

At the opposite, streaming activities pass items between each other anytime while executing.

Modelling is advantage: obliges you to question yourself about the system you want to create

↳ Engineer build and design first. We don't want to represent reality.

FUNDAMENTALS OF INFORMATIONS SYSTEMS

INTEROPERABILITY

why Interoperability:

"ability of two of more systems of adapt on a same kind of file, format, protocol."

Examples are in amazon marketplace → payment system → payment → shipment → customer center

All modern systems are modules that work with other services (by other information systems too)

	Syntactical	Semantics	Organizational
Exchange	common formats like XML, JSON, YAML, SQL...	Schema matching and mapping; ontologies	Message exchange
Integrate	(from more to unique) SQL/XML standard, native XML DBS, REST	EdgeTable, Shredding, schema & data integration	Correlation & choreography
Orchestrate	BPMN, Petri Net, workflows nets	Verification: task and worklist design; service invocation	choreography

↳ I want to have UNIQUE POINT on where to find! Single Point of Orchestration

Integration is the MAIN issue

Integration

POSSIBLE in different Application domain levels:

UI ORIENTED
- GUI Integration : done through Portals and Mashups
  -should be done only when other options not necessary-

MESSAGE ORIENTED
- Process Integration : process engines
- Function Integration : Message-passing, publish-subscribe

QUERY BASED
- Data-Information Integration: Is query based, no procedure to carry on, but just data

XML Foundation

XML (Extensible Markup Language) is an important and standard for exchange of data, usually web-based application.
It's comprehended in trees, with elements containing texts, in which that can be added some attributes.

It supports the addition of generators, parsers etc:
- NAMESPACES
- QUERIES (e.g. Xpath to query data)
- TRANSFORMATION (XSLT)

Another common scenario is the creation of standards to describe domain specific informations utilizing XML files. Creates common data structures:

<? xml version="1.0" encoding="utf-8" ?>
<? xml-stylesheet .. ?>
<! DOCTYPE book SYSTEM "book.dtd">
< book name="interop">
chapter number="1".

processing instruction

DOCTYPE definition
Root element
element
attribute
entity reference
start tag
end tag
comment

Well formed: doc is written to syntax in XML

VALID: document is compliant with the schema

XML is a good way to exchange data!

For example, we may want to export database information from a relational DB. We can translate it through XML as a way to exchange data

RPC can be exposed to internet with standard.
JRMI is a ORPC.
Problems with security! Difficult to work over firewalls:
invent XML-based protocol that could work over HTTP.

SOAP

inter-application communication through HTTP, marshalled over XML to encode.
we have a SOAP envelop (standerdized)
with inside it a SOAP body, containing request.

Response is same.
Everything is standardized to use HTTP POST.

Adv:
- HTTP reliant
- portable & interoperable
- universally accepted

Disadv:
- Too much reliane over HTTP
- stateless
- Serialization BY VALUE, not by reference

2 communication styles:
- RPC
- Documents → used maybe more as a request-reply of docs.

SERVICE DESCRIPTION

Way for clients to discover interfaces, XML that provides interfaces using WSDL (Web Service Description Language).

(kind like Interface Definition Language - IDL)

It's a contract between requestor and replier.
2 different parts:
- Definition → how to use it
- Implementation → where to find it

RESTful Services

REpresentational State Transfer is a paradigm, references to simple application interfaces transmitting data over HTTP without any additional layers.

↳ Requires specific architectural styles.

Metaphorically, we have URIs as nouns and HTTP commands as verbs. Has state, meaning having application/session state.

Methods have been previously defined! - kind of RPC -

Example

"getStockPrice" cannot be linked to a verb GET POST PUT

if we change it to "currentStockPrice" it does!

GET	"currentStockPrice"	→ gets
PUT	"currentStockPrice"	→ updates
DELETE	"currentStockPrice"	→ deletes
POST	"currentStockPrice"	→ adds

Examples.

management of orders and customers

CUSTOMERS	ORDERS
/customers	/orders
GET	GET list all orders
POST	POST adds order
/customer/{id}	/order/{id}
GET	GET get detail
PUT	PUT update order
POST	POST add order
DELETE	DELETE deletes
/customer/{id}/orders	
GET get orders from cust	
POST add order	

the REST API linked to an actual interface with getter/setter paradigm.

e-Services

Application accessible via the web, that provides functionality to businesses or individuals.
Another definition, more popular and mature, defines a component as an e-service following 3 points:

- openness: independent, as much as possible, of specific platforms and computing paradigm
- developed mainly for inter-organization applications: used mostly between DIFFERENT organization / application
- easily composable: well founded, so that it can be used as building block for other applications.

WEB-SERVICE

W3C standardized also this term: sw system identified by URI, whose public interfaces and bindings are defined using XML.
Definition is discoverable by other systems, and used mostly by other systems.

↳ XML because REST wasn't still invented.

for building an e-service a designer may need to implement many web services!

Process Orchestration

We need to orchestrate everything so that every information acquired using interoperability is correctly used

↳ [Can be done by modelling it through a sysML diagram.]

Internally, other than having diagrams to represent orchestration, usually YAML or XML gets used to actually orchestrate.

we need to have - for project.
- containers
- orcherstration

} docker-compose.yml

Petri Net is a mathematical semantic model to describe orchestration. Useful for formal verification.

DOMAIN DRIVEN DESIGN & MICROSERVICES

We have small, INDIPENDENT, modules containerized, that can communicate between each other.
Microservices are deployed per CONTAINER! (1:1)

Idea is agile methods and devOPS, in which we deliver small chunks of functionality, derived also with user stories.

It's a MODEL-DRIVEN design approach, based on collection of principles and pattern that helps crafting elegant system.
A DDD is a way to set set of priorities to deal with complicated domains

A domain is a sphere of knowledge or activity. What an organization does and the world it does it in.

A model is a system of abstractions that describes selected aspects of a domain and ignores extraneous details. It's a distilled version of a domain
It should model EVENTS-OF-LIFE.
↳ methodology programming independent.

IMPLEMENTATION <=> REALITY

It's ubiquitous language, in which models are explored in collaboration between domain and software practicioners.

FOCUSES on the core domain, that bounds models and implementation together.

> A change in language means a change in model and code.

Model expression is agnostic to the modelling language, and it's expressed in the code.

BIG DESIGN UPFRONT

When building a model with this way, at then they'll be created "clusters of concept" similar to one another.

Ex e-commerce

- promotion
- shipping
- product building
- inventory
- ...

Bounded contexts clusterizes concepts related to one another, (that they'll be candidates for microservice building)

Domain can be generalized into 3:

- CORE
- SUPPORTING
- GENERIC

↓ first → last

prioritization like the product owner does in SCRUM

LAYERS

(presentation layer)		(app. logic)	(DB & middware)
USER INTERFACE	APPLICATION	DOMAIN	INFRASTRUCTURE

the infrastructure is a supporting library for all other layers.
the domain layer contains the idea of the domain. Persistence is infrastructures job
The application layer is an orchestration that coordinates information presentation.

Concepts will be modelled as [...]

- Entities
- Value objects
- Aggregates
- Services
- Association
- Factories
- Repositories
- Modules
- Context mapping.

3/4 practical microservice patterns.