More **9.3.2 Architecture**
Also see 10.3.1 A Brief Taxonomy of Architectural Styles
DB-Centered Arch (AKA Data-Centered)
DB Major Data Parts (**RDB** – no Objects – **SQL**)

> RDB from "Relational Algebra", by Edgar Codd 1970 [nox]
> o- **Tremendously simpler** than prior DB kinds
> o- Took over by late 1970s
> o- Initial RDB "languages", mostly gone: Sequel, SQL, Postgres
> SQL Today (pronouced S.Q.L or "Sequel")

o-many **CRUD** clients (Create, Read, Update, and Delete data)
RDB must be Reliable/Resilient
RDB typically Distributed – DB is split up = diff parts of data in diff PU (Processing Unit) locations
Issues:
Redundancy == duplicate data copies on diff PUs, but local copy changes **must be mirrored** on other PUs
Consistency == all local copy changes **are mirrored** on other PUs
Consistency time delay == how long does it take to get a local data change mirrored?
Atomic (multi-part) transactions (all parts win or all parts lose) = complicates Redundancy, Consistency
Single-computer, multiple-disks (Redundancy) – typically uses "RAID-5" tech
o- Destroy any one (of 5 disks) and no loss of data – similar to doing parity-based Error Correction
EX: MySQL, SQLite (freeware)

Giant DB/Cloud
o- Too much for single-CPU processing – only Distributed or Parallel or both
NoSQL (generic term) for Giant (non-RDB) DBs – cuz it simplifies scaling up – (but scales down, too)
o- **Key-Val DB** (AKA Map/Dictionary)
EX: Redis, Memcached
o- **Document** DB – Hierarchical Structured Data: JSON, XML docs – large and small docs
EX: MongoDB
o- **Graph** DB – directed-edge/arrow relationships between data-objects (no methods)
EX: Neo4j
Issues:
ACID: Atomicity, Consistency, Isolation, Durability
Atomicity: all or nothing rule (parts of transaction) – needed for multi-part transactions
Consistency: only valid data in DB (w.r.t. Constraints) with diff data parts in diff PU locations
Isolation: seq of overlapping xtns can't interfere w each other – regardless of time-order of their sub-parts
o- Diff parts of each transaction can overlap in timing
Durability: DB atomicity even if crash during transaction (during its several sub-tasks)
o- you can pull the power plug & the DB is still safe – all the atomic transactions that completed on reboot
BASE: [**Nox**] (Wordplay on "ACID" from Chemistry)
Basic Availability (AKA local availability) – when can you get (completed) data?
Soft-state (AKA inconsistent for newest local changes – until they are all duplicated properly)
Eventual consistency (AKA consistent for older local changes – cuz they've been duplicated)
o- Badly chosen expressions (hard to remember)

CAP 'theorem': can't have **Consistency**, **Availability**, and **Partition** tolerance (in the cloud),
→ **you have to settle for two out of three.**
(Eric Brewer) [**Nox**]
Partition: split up == distributed data parts on diff PUs
Consistency: == all processors see/know same thing: How?
By no (immediate) **Availability**: (by delaying local access till remote data change is here too)
By no **Partition**: (by having no remote data & PUs) – IE, no cloud – just local data processing
Availability: distant changes are available "immediately"
By delaying local access till remote stuff is visible

o9.2.1 Software Quality Guidelines and Attributes p312
- o- **Non-Fcnl Reqts** (AKA **Quality Reqts**) (AKA the "**ilities**") – not Doing it, but How it gets done
- o- NB, Fcnl Reqts are actions your program takes; Non-Fcnl Reqts are how it gets the job done

There have been a lot of Quality Reqts Checklists – over the last 5 decades
FURPS (learn the acronym)
- o- **F-**unction**ality** (Actually, NOT a **Non-Fcnl** Reqt; included by "committee")
- o- **U-**sab**ility** (Doesn't make the user work hard [**UI Rule #1**])
- o- **R-**eliab**ility** (~ always works when the user needs it)
- o- **P-**erformance (AKA Efficiency) (**fast**, small, frugal)
- o- **S-**upportab**ility** (AKA Maintainab**ility**) (easy to **find/fix/extend**)

Quality Concepts
ISO 9126 S/W Quality Factors [**nox**]
- o-- NB, **ISO** == Internation Stds Org; **ANSI** == American National Stds Institute

FURPS + Portab**ility** [**nox**] (to another CPU, OS, Framework, Platform, Language)
- o- Was superseded a few years ago (now with 30+ ilities, and counting) (**ISO 25010**) [**nox**]

oXX. (S/W) Review Techniques p 325
- o- *"The later you find a mistake/defect, the costlier the fix."*

Kinds, reviews for coding: (usuallly by team mgmt, of junior coders)
- o- **desk check**: informal, **by yourself**, go over the code, try to find mistakes (design & code)
- o- **walk-through**: formal, go over the code w/ 3-6 onlookers, ("Action Items") task issues to be fixed
  - o-- time-consuming (3-6 staff for, say, an hour), so expensive
- o- **inspection** (Gilb 1993 nox): formal, go over randomized (say ~ 20%) (statistical) sample of code, pass if no issues in sample – (*) saves time

** Some people use "walk-through" and "inspection" as **synonyms**
- *- XP's (1993) **pair pgmg** has built-in "continuous review" (two heads at one desk/screen/kybd)
  - o-- Key Pbm with Pair Pgmg == Mgmt: paying two people to do the work of one?

- *- "**Lessons Learned**" **after project** (AKA Post-mortem analysis)
  - o-- to improve processes for next project
- o- **Agile** "**retrospective**", after each "sprint" (AKA devel period)
  - o-- "velocity" (Team feature dev "speed", in story pts per time) & quality, stats
  - o-- story points, estim: (should have error bars), trend line
  - o-- workflow hiccups (what went wrong)
  - o-- umbrella/admin issues // outside S/W dev team
  - o-- big **tech debt** issues (**Tech debt** is stuff you saw that **needs to be cleanup**, but wasn't)

oXX. SW Quality Assurance p 339
SQA == **SW Quality Assurance** (often a dept in a big S/W organization)
Goal: that stds/policies/SW-Dev-procedures are actually followed
Audits: **Reviews** are called "Audits"
Testing: ** To find errors (Mindset: "**Break it**")
SQA checks test planning, test execution processes, and test result docs, & coding stds being followed, ...

(*) **Improve via measurement**

*"To measure is to know." [**nox**]*

*"If you can not measure it, you can not improve it." [**nox**]*

Q: What are you not measuring?

Q: Can your measurements **predict reliably**? (most S/W measurements don't)

o- We collect measurements to predict the future (with error bars)

Q: What % of devr time do your measurements **take away from productivity**? (a reason for doing less meas'g)

Q: Is it simple to measure, or costly?

Std things to measure:

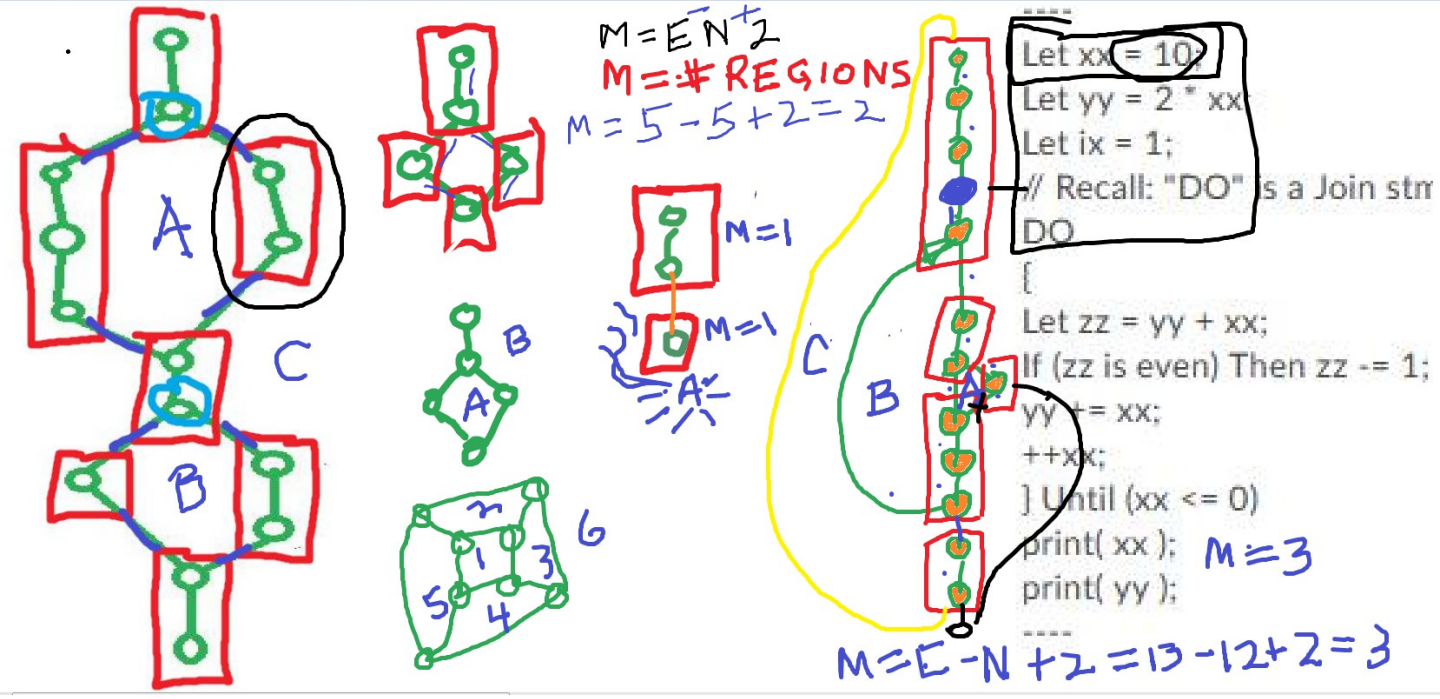
LOC/SLOC (KLOC/KSLOC) = **Lines of Code**, or Source Lines of Code, K = thousand
(usually has big error bars – 3x?)

Function Points (> 4 params; need 100s of hours to learn to do it well; used for up-front BUFD estimates)
"With 4 parameters, I can model an elephant; with a 5th parameter, I can make the elephant dance."
– John von Neumann (the Martian)

Cyclomatic Complexity (how many cycles/loops are in the program; done for a "unit" of code, or a fcn)
o-- "Unit" of code = one person, dev from 1 to 20 days (3 weeks)
o- Works for small fcns, mostly
o- Used as a measure of how complicated a Unit's design is – in a code review

Cyclomatic Cplxty

Created by McCabe, henced denoted "M"
o- Normal counting via "MEN2" → **M = E – N + 2**
N = nodes (AKA straight-line code segments; AKA "Basic Blocks")
o-- Node **ends** with A) a **branch** stmt (2+ ways out), or B) a "**return/exit**" stmt
o-- Node **starts** with A) an "**entry point**" (first) stmt or B) a "**fan-in**" or "**join**" stmt
E = edges between "nodes"
o- Alt: [what I use] Count the number of planar graph regions – you can "eyeball" it
o- Loop back up to a node is just an edge splitting at the loop bottom and joining at the loop top
o- Personal variation – Allow well-formed switches as one loop/region



Q: What's the Cyclomatic Complexity of this piece of code?

```
if (edges == 0 || subsets == 0) print("Warn Conga #1");
gInsertions += np;
if (hms == 0) {
..hms = 1;
..unsigned long n = np;
..while (n > 2) {
....n /= 2;
....hms++;
....}
..}
if (hms == 1) print("Warn Conga #2");
if (hms > MaxCongaSubsets) print("Warn Conga #3");
how_many_subsets = hms;
subset_sizes.Allocate(hms);
```