o25.6.1 Software Sizing p511 (8th ed in Ch 33)

Based on: **4 Keys** –

o1- **Code Size** – the degree to which you have properly estimated the size of the product to be built
o-- Estimated in LOC (by summing over all estimated sub-tasks) or in FP (Function Points)
o-- LOC for the same function can easily vary by a factor of 2

FP Cons – Pressman & Maxim, p481, 23.7 Software Measurement
Opponents claim that the method requires some "sleight of hand" in that computation is based on **subjective** rather than objective data, that counts of the information domain (and other dimensions) can be **difficult to collect after the fact**, and that FP has **no direct physical meaning**—it's just a number.

o2- **Effort from Size** – the ability to translate the Size estimate into human effort, calendar time, and dollars
o-- Historical Baseline – requires "reliable software metrics from past projects"

Pressman & Maxim, p481, 23.7 Software Measurement
However, to use LOC and FP for estimation (Chapter 25), an **historical baseline** of information must be established. It is this historical data that over time will let you **judge the value** [ie, accuracy] of a particular metric on future projects.

o3- **Team Abilities** – the degree to which the project plan reflects the abilities of the software team
o-- You must "know" your people well

Steve McConnell, p568, in Making Software: What Really Works, by Oram, 2010
o- Author of textbooks – eg, Code Complete, Rapid Development, and Software Estimation
The original study that found **huge variations in individual programming productivity** was conducted in the late 1960s by Sackman, Erikson, and Grant [1968]. They studied professional programmers with an average of **7 years' experience** and found that the ratio of initial coding time between the **best and worst** programmers was about **20 to 1**; the ratio of debugging times over **25 to 1**; of program size **5 to 1**; and of program execution speed about **10 to 1**. They found **no relationship** between a programmer's **amount of experience and code quality or productivity**. …

In the years since the original study, the general finding that **"There are order-of-magnitude differences among programmers"** has been confirmed by many other studies of professional programmers [Curtis 1981], [Mills 1983], [DeMarco and Lister 1985], [Curtis et al. 1986], [Card 1987], [Boehm and Papaccio 1988], [Valett and McGarry 1989], [Boehm et al. 2000]. …

When I was working at the Boeing Company in the mid-1980s, one project with about **80** programmers was at **risk of missing** a critical deadline. The project was critical to Boeing, and so they moved most of the 80 people off that project and brought in **one guy** who finished all the coding and delivered the software **on time**.

o4- **Stable Reqts** – stability/unchanging of product requirements and the (development) environment support
o-- Does it matter if the Reqts are Stable?

Steve McConnell (p58 in Software Estimation: Demystifying the **Black Art**, 2006)
"potential differences in how **a single feature** is specified, designed, and implemented can introduce cumulative differences
of a **100x or more** in implementation time"

(*)* One Feature → 100x impl time variance from an initial Estimate == very large Error Bars
o-- Agile M.O. assumes **Reqts can not be stable**

Empirical models (eg COCOMO III (Boehm), or Function Points)

Cons:

o- needs good baseline for knobs settings (AKA parameter values)

o- poor sensitivity → change knobs slightly, but get big (non-linear) change in results?

o- too many knobs to be trustworthy – famous quote by John von Neumann, via Fermi via Dyson

> Freeman Dyson being schooled on model parameters by Enrico Fermi:
>
> In desperation I asked Fermi whether he was not impressed by the agreement between our calculated numbers and his measured numbers.
>
> He replied, "How many arbitrary parameters did you use for your calculations?"
>
> I thought for a moment about our cut-off procedures and said, "Four."
>
> He said,
> "I remember my friend Johnny von Neumann used to say,
> with four parameters I can fit an elephant, ["fit" means "make a model for"]
> and with five I can make him wiggle his trunk."
>
> With that, the conversation was over.

More on Team Ability Estimates

> **Steve McConnell**, p568, in Making Software: What Really Works, by Oram, 2010
>
> [A Tale of Two Spreadsheets]
>
> One specific data point is the **difference in productivity** between Lotus 123 version 3 and Microsoft Excel 3.0. Both were desktop spreadsheet applications completed in the 1989–1990 timeframe. Finding cases in which two companies publish data on such similar projects is rare, which makes this head-to-head comparison especially interesting. The results of these two projects were as follows: Excel took **50 staff years** to produce 649,000 lines of code ... Lotus 123 took **260 staff years** to produce 400,000 lines of code … Excel's team produced about 13,000 lines of code per staff year. Lotus's team produced 1,500 lines of code per staff year. The difference in productivity between the two teams was **more than a factor of eight**...
>
> Both Lotus and Microsoft were in a position to **recruit top talent** for their projects.

(*)* Management style trumps individual team member abilities

Combine Multiple Estimates & **Guess** Distribution **Mean (but not Variance/Std Deviation)**

Simple 3-point weighted approximation to a **Bell Curve** (AKA Gaussian, Normal Distribution Curve):

o1- Do 3 Estims: S=**Optimistic**, M=Normal/middling, L=**Pessimistic**

o2- Combo Estim of the Mean: **Size = (S + 4*M + L)/6** – very rough approx of a bell curve – Eqn 25.3 p519

o-- Often the L is much larger than the S is smaller – lop-sided toward the L side

o-- Take (L – S)/2 as 1 std deviation from the mean; hence, Mean + (L – S) "could" be the 95% win Pbb

Con:

o- Very rough

o- This isn't a good explanation for Mgmt

o McConnell – Estimate vs Commitment

(*)* Always add a likelihood/Probability % to your estimate (eg, "80% of the time") when giving it to Mgmt

o- Mgmt think – Estimate == a Commitment & a Guarantee to doing the job in that time with that budget

o25.7 Project Scheduling 520 (8th ed Ch 34)
o25.7.1 Basic Principles 521

1. **Compartmentalize**: Make the WBS
2. **Interdependence**: Task B can't start till Task A is done
3. **Time-Allocation (to tasks)**: duration, start-end; LOE (Level of Effort) (eg 50% time)
4. **Effort validation** (AKA **Max #Tasks in Parallel)**: based on **who is available** to work them?
5. **Defined Responsibilites**: who gets each task, for scheduling
6. **Defined Outcomes**: What proves a task is completed?
7. **Defined Milestones**: Date to review/approve a **cluster of completed tasks** (eg, for a feature)

o25.7.2 People & Effort p 522 (8th ed 34.2.2)
o- When fallen behind, adding more people usually doesn't help.
o- **Brooke's Law**: "Adding manpower to a late software project makes it later."
Why?
o1. Up-to-Speed Time, Pgm – getting the added people familiar with the pgm being developed
o2. Dev-Teacher Time – dev'rs taking time away from working tasks to teach the added people
o3. Comm Time Increase: usually increases by N^2, where N = #staff – but heavily mitigated by Standups
o4. Up-to-Speed Time, People – who does/knows what, who's easy/hard to talk with

Lab: CF 343-p3-vcs-merge-crc-arch-draft.pdf

from Brooks' Silver Bullet paper
o-**5.3 Incremental Dev [and Review by Cust]: Grow, don't Build**
"The **morale effects** are startling. Enthusiasm jumps when there is
a running system, even a simple one."
"One always has, at every stage in the process, a working system."
Recall: **Morale is the Most Important Thing** in S/W Development

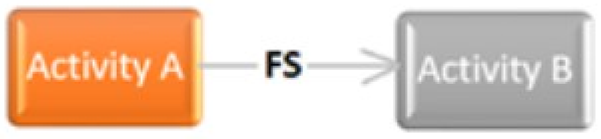
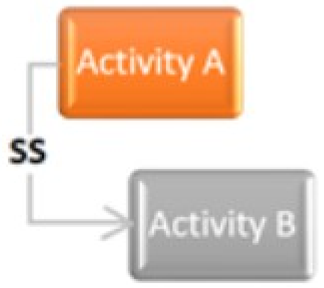
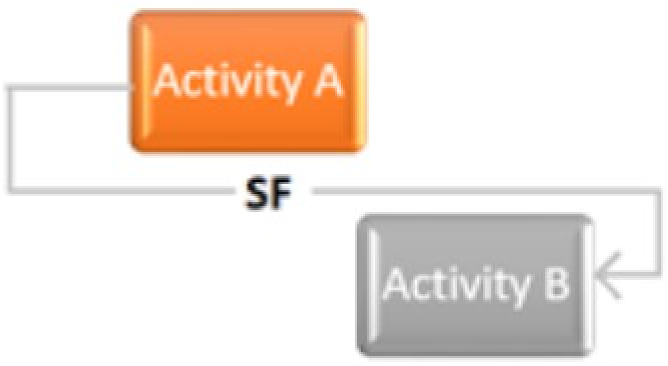
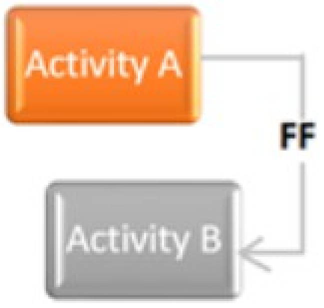
o SW Proj Scheduling
Time Concepts: **Lead, Lag, Slack/Float**
Informally:

o-- Kid Task – depends on its mom task(s) finishing first (DAG: graph parent-child node stuff)

o- **Earliest start time (ES) -** The earliest time, a task can start once previous mom tasks are over (or started)
o- **Earliest finish time (EF) -** This would be (ES + task duration) for the kid (or mom) task
o- **Latest finish time (LF) -** The latest time a task can finish without delaying the project (or a kid task)
o-- Alt: without delaying any of its kid tasks
o- **Latest start time (LS) -** This would be (== LF - task duration)

Inter-Task Constraint Relationships:

o- **Parallel** (ie, no constraint relationships between tasks)

o- Finish-to-Start (FS) – S/W Engr Normal  *Finish to Start*	o- Start-to-Start (SS) == Req'd delay for kid  *Start to Start*
o- Start-to-Finish (SF) == kid has a hard finish time/date  *Start to Finish*	o- Finish-to-Finish (FS)  *Finish to Finish*

Lag Time: Intended Delay
 o- FS with a Lag: mom.end (+delay) to kid.start // EX kid coat waits for mom-paint coat to dry
 o- SS with a Lag: mom.start to kid.start // EX do complex rebar work before setting forms around it
 o- FF with a Lag: mom.end to kid.end // EX after concrete pour is done, wait 10 days to remove forms

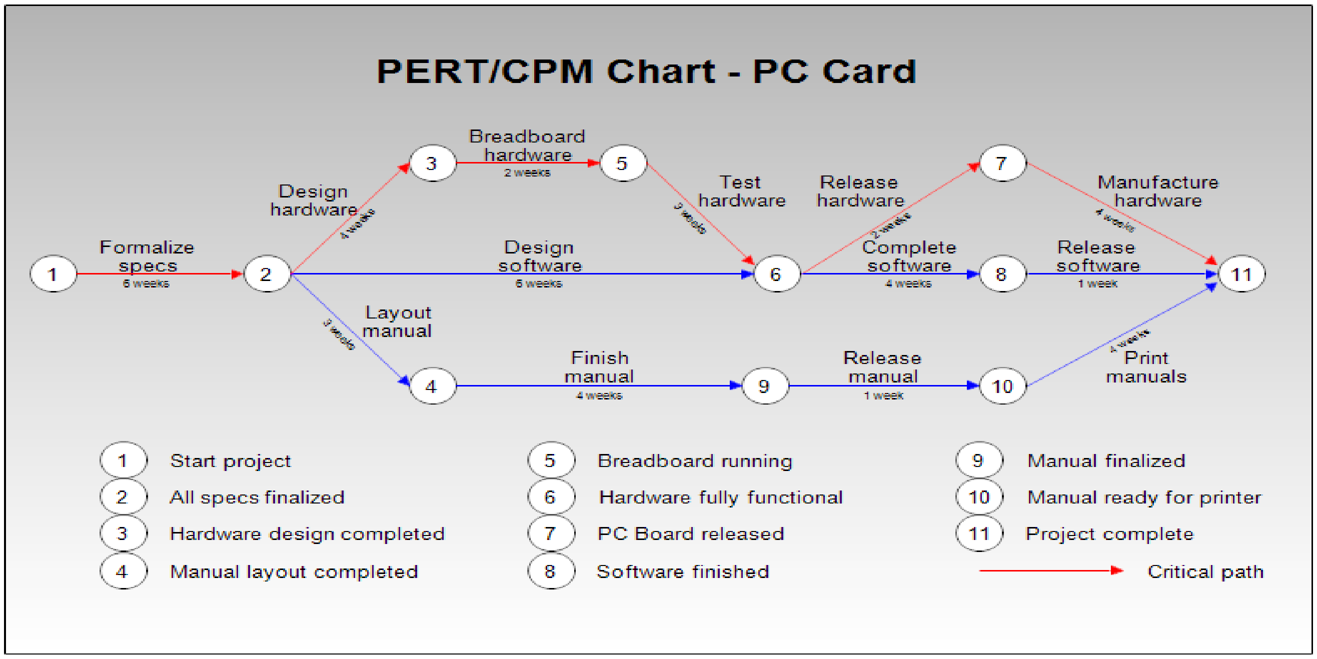
Lag Time: Unintended Delay
 o- FS with Slack time: kid could start earlier (mom's already finished) but there is nobody available
  o-- Short (and usually reasonable) staffing size
  o-- Requires S/W domain expert (who is not yet available) – eg, SQL (DBA – DB "Administrator")

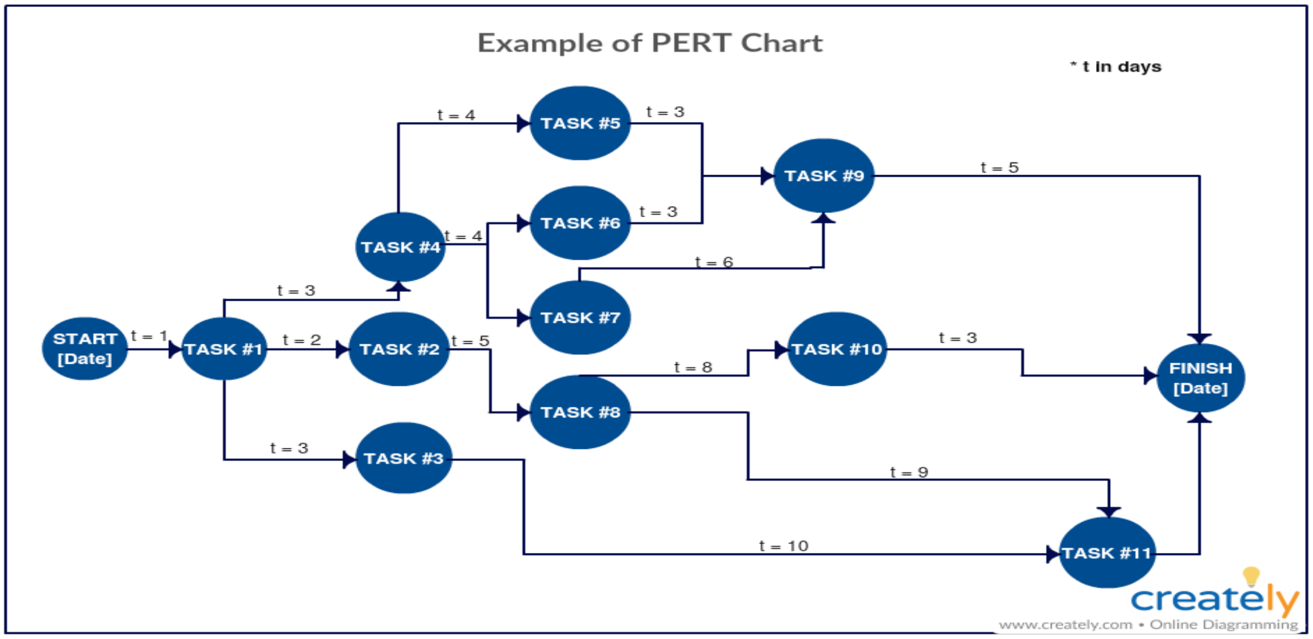
Lead Time: Interleaved:
 o- From kid.start to mom.end
  o-- Apparently, kid only depends on an "earlier part" of mom – maybe mom could be further split
 o- Equiv to splitting mom into non-dep + dep sub-tasks
 o- Alt: Equiv to splitting kid into non-dep + dep sub-tasks
 EX: Start painting-walls (on all drywall sections finished), don't wait till all drywall is up
  o-- Means we DON'T have to create a blizzard of small drywall-next-4-feet tasks & their paint tasks

Slack (AKA "free float" or just "**Float**")
 o- kid-**start movable** in interval from mom.end to kid.start
  o-- Like **kid floating in a bathtub** – mom.end **is left side** of tub & grandkid.start **is right side**
 o- But **kid can't push mom or grandkid tasks**

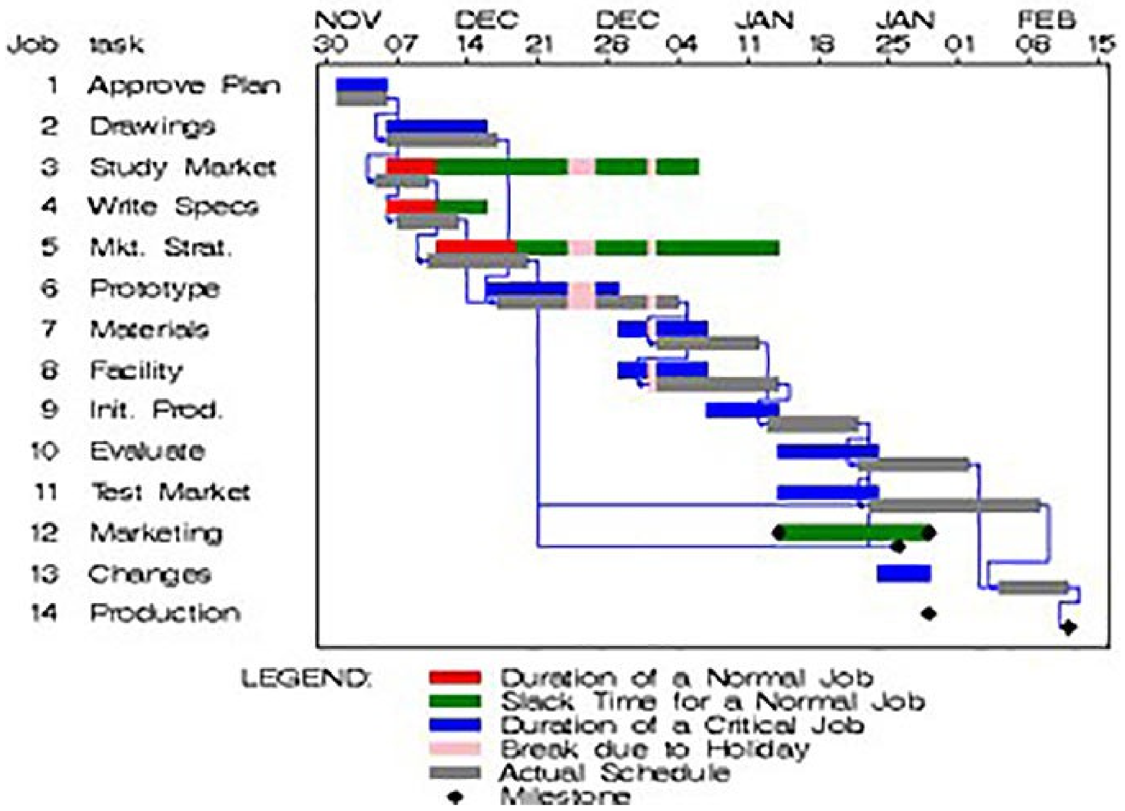
Represent task dependencies
 Use a **PERT chart mech** to generate a **graph**
 o- based on **mom-to-kid inter-task dependencies** (for S/W, normally FS dependencies)
 o- and then use a CPM (**Critical Path** Method) mech to find **minimum project timeline**
  o- Critical Path is required – if you believe the inter-task dependencies you've laid out

PERT Chart: (Pgm Eval and Review Technique, nox) (Charts from Wikipedia images)
AOA Style: "**Activity on Arrow**"
edge == duration (in Graph Thy, called "edge weights")
o- Task Name on edge, Sometimes
node == milestone (AKA Task Completion Event)
Alt: **AON**: Activity on Node :: node == task+duration :: edge == milestone & kid task
Big Question: What is the minimum time to complete the project, given the Task Times?
(*) AKA the "**Critical Path**" – **tasks on minimum-time path (from start to finish)**
**** There is NO SLACK/FLOAT on the Critical Path**

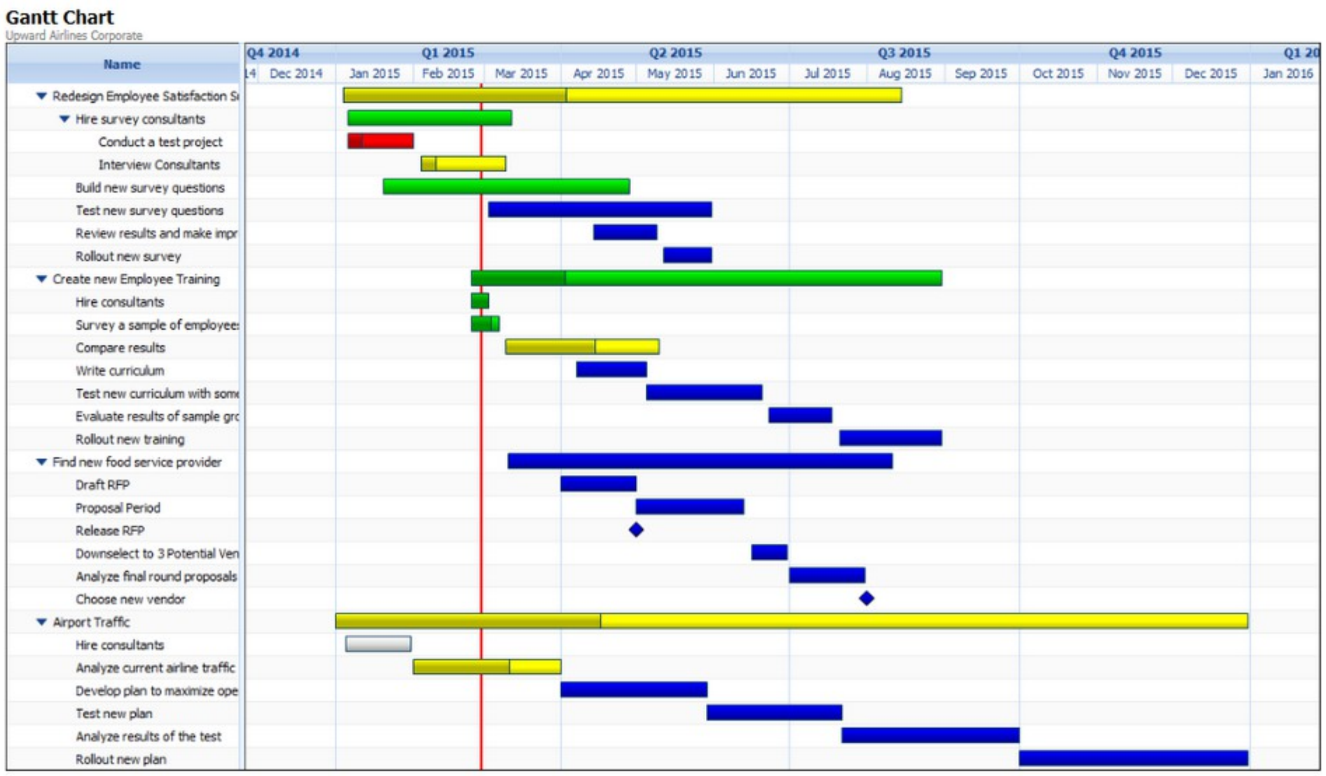


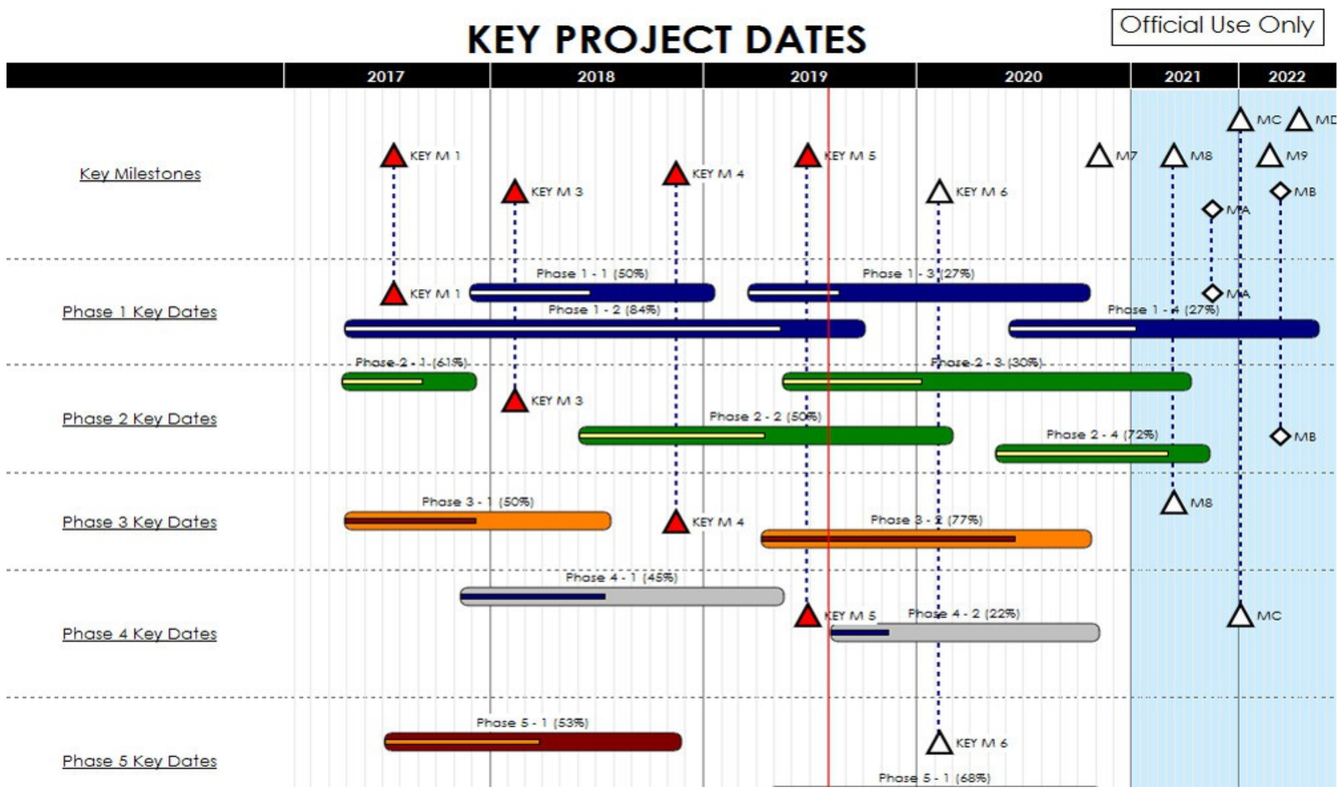


Gantt Chart + Pert dependencies



Gantt Chart


Gantt Chart
Upward Airlines Corporate
Name
Q4 2014
Q1 2015
Q2 2015
Q3 2015
Q4 2015
Q1 20
14
Dec 2014
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
Dec 2015
Jan 2016
Redesign Employee Satisfaction S
Hire survey consultants
Conduct a test project
Interview Consultants
Build new survey questions
Test new survey questions
Review results and make impr
Rollout new survey
Create new Employee Training
Hire consultants
Survey a sample of employee
Compare results
Write curriculum
Test new curriculum with some
Evaluate results of sample gr
Rollout new training
Find new food service provider
Draft RFP
Proposal Period
Release RFP
Downselect to 3 Potential Ven
Analyze final round proposals
Choose new vendor
Airport Traffic
Hire consultants
Analyze current airline traffic
Develop plan to maximize ope
Test new plan
Analyze results of the test
Rollout new plan


KEY PROJECT DATES
Official Use Only
2017
2018
2019
2020
2021
2022
Key Milestones
KEY M 1
KEY M 3
KEY M 4
KEY M 5
KEY M 6
M7
M8
M9
MA
MB
MC
MD
Phase 1 Key Dates
Phase 1 - 1 (50%)
Phase 1 - 2 (84%)
Phase 1 - 3 (27%)
Phase 1 - 4 (27%)
Phase 2 Key Dates
Phase 2 - 1 (61%)
Phase 2 - 2 (50%)
Phase 2 - 3 (30%)
Phase 2 - 4 (72%)
Phase 3 Key Dates
Phase 3 - 1 (50%)
Phase 3 - 2 (77%)
Phase 4 Key Dates
Phase 4 - 1 (45%)
Phase 4 - 2 (22%)
Phase 5 Key Dates
Phase 5 - 1 (53%)
Phase 5 - 1 (68%)

o26 Risk Mgmt p 532

RMMM [nox] == **Risk**
o- **Mitigation** = lessen impact (being proactive)
o- **Monitoring** = Tracking = Monitoring – hold mtgs, update risks
o- **Mgmt** = (create new mitigation tasks) – mitigation means making a bad situation less bad

Reactive vs Proactive
o- **Reactive**: Wait until event (bad thing) occurs, then "think of something"
o- **Proactive**: Think of how to deal with possible upcoming event beforehand
o-- Risk Mitigation Plan + Monitoring (eg, periodic meetings)

Kinds of Risks:
o1. Project Risks: Impact: schedule slip, cost overrun
o-- Pbm Sources: (changes to) budget, schedule, staffing, resources, **reqts**
o2. Tech Risks: Impact: SW quality, on-time delivery
o-- Pbm Sources: design, impl, interfacing to other systems, verification, maintenance, spec ambiguity (spec == formal written requirements), inferior tools, tool obsolescence, bleeding edge tech that you decided to use on this project
o3. Biz Risks: Impact: up to stopping project – or even product line
o- Mktg/Marketing risk: cust/user hates it
o- Strategic risk: bad business fit (eg, fratricide – it kills off sales of your older “cash cow”)
o- Sales risk: confusing to potential buyers/custs, or confusing to sales representatives (Sales Reps)
o- Mgmt risk: C-level (Company executives level) changes in focus and/or people
o-- Historically, this has been an issue for adopting Agile Dev M.O.
– the point of Cockburn & Highsmith's observation that Politics wins over (strong) People
o- Budget risk: C-level reduces budget; or refuses extra funding to extend project; can't win over customer

Assessing Overall Project Risk
o **Key Questions**, in order of importance – “**No**” answer means it **needs tracking**
o1. Top SW & Cust mgrs formally committed to support the project? (enthusiastic, or at least hopeful)
o2. Users enthusiastic to getting their hands product?
o3. Reqts understood by dev team & custs/users? (usually can't know this till part way into project dev)
o4. Cust/Users involved in reqts dev? – (Actually, users involved in reqts + CRC modeling & hand-sim?)
o5. Users realistic about product? – nothing fanciful is expected?
o6. Scope stable? – Scope == set of Features == set of Requirements → this is very rare
o7. Dev team have req'd skills? – especially if you expect to hire some specialty-skill programmers
o8. Dev team has tech experience in the S/W area (maturity level experience) to be used?
o- (With respect to) extra knowledge not written down
o9. Staff size adequate? -- Key people (specialty S/W areas)?

Risk “Projection” (AKA Risk Estimation)
4 Steps
o1. **Pbb**: Establish pbb scale (Pressman & Maxim rec's 10% increments; but it's far too detailed)
o-- Instead, use tee-shirt sizes for Pbb: small, medium & large – more appropriate for accuracy
o2. **ID consequences** of risk event: What happens?
o3. **Estimate event** cost or **impact**: (see next section)
o4. **Assess estimation accuracy**, if possible (add Error bars – plus/minus one tee-shirt size)
o-- Crowd-source to get more varied viewpoints on the Risk

Risk Scale Class – very rough categorization – These should reflect Mgmt “Concern”
[Pressman & Maxim recommended scale – **nox**]
1=**Catastrophic** – main purpose failure – system basically unusable
2=**Critical** – reduced capability – significant loss of some major functionality
3=**Marginal** – failure of some minor functionality (AKA “nice-to-haves”) – painful to users
o-- EG, like going from laptop word-processor program back to a typewriter – doable, but painful
4=**Negligible** – failure causes **annoying workaround** – all user needs except reduced quality
o-- EG, like going from automatic zip-code city name entry to looking up the city by zip manually

Assessing Risk Impact
Risk Exposure = RE = **Risk Severity**
RE = Pbb * Cost // Pbb_of_Occur * Cost_of_Event
o-- **Cost is the same as Impact**

Mgmt Steps:
ID (Identify) Risks
o1. ID Risk: summary, tag-line, title, index number (#X)
[o2. ID Risk Scale Class – recommend using names, not numbers (eg, “ Marginal”) – usually ignore this]
o3. ID its Pbb: (Lg,Md,Sm) = (3..1) – very rough estimates (only kind you likely can likely do)
o4. ID its occurrence Impact or Cost: Lg,Md,Sm (3..1) – very rough estimates

Select Risks to Track
o1. Calc: Severity = **RE = Pbb * Cost** == (9..1) == (LL..MM..SS)
o-- Is No 5 or 7, prime – only RE is in (9, 6, 4, 3, 2, 1)
o-- Throw out (3,2,1) cost risks – they are too tiny to care about
o-- Maybe keep RE=3 → Pbb=1 but Cost=3
o2. Keep: (9..4) to track 3 values = (9,6,4)
o-- NB, (9,6,4) aren't hard numbers – R=6 isn't 150% of RE=4 – they are rough relative numbers
o-- This type of estimate is **very subjective**
** On large projs, tracking 40+ risks not unreasonable – typically never track 200+ risks

ID Mitigations, if any
o- For Risk ID #X
o- Reduces Pbb?
o- Reduces Impact?
o- Benefit of Mitigation – Reduces Severity (RE eqn) to what?
o- Cost of Mitigation? – is it worth doing?
o- When can it be applied?
o- Who needs to be involved in performing the Mitigation?

Mgmt: Track Risks – Periodic Mtgs (eg, weekly, fortnightly, monthly)
o- Write Risks down
o-- **Short summary** sheet (one risk per line): ID #X, title, class (Project,Tech,Mgmt), Pbb, Impact, RE, Has-mitigations (Y/N)? Mitigation tasks in progress (Y/N)? – 8 columns
o-- **Long form** Risk entries in a booklet: add tag-line/summary, mitigations (potential & tasked)
o- Re-evaluation – Weekly/fortnightly/monthly (hopefully quick – < 30 mins per meeting)
o-- Risk eval “committee”, usually 3-4 people

o-- Existing risk active (tasked) mitigations on track
EX: Contacted backup supplier
EX: Updated As-Built arch doc (eg fancy UML printout) this month
EX: Cross-area training/tasking is up to date – lose the SQL expert, but you have a backup
o-- New risks? – add short line & long page – calc RE – guess mitigation if any
o-- Changed Pbb?
o-- Changed Impact?
o-- Changed Mitigation? (eg, thought up some new (reasonable) mitigations)
EX: Alice (on diff proj) was backup for Bob, but Alice quit – need a new backup
EX: Hiring cost for newbies just lowered because extra training docs available

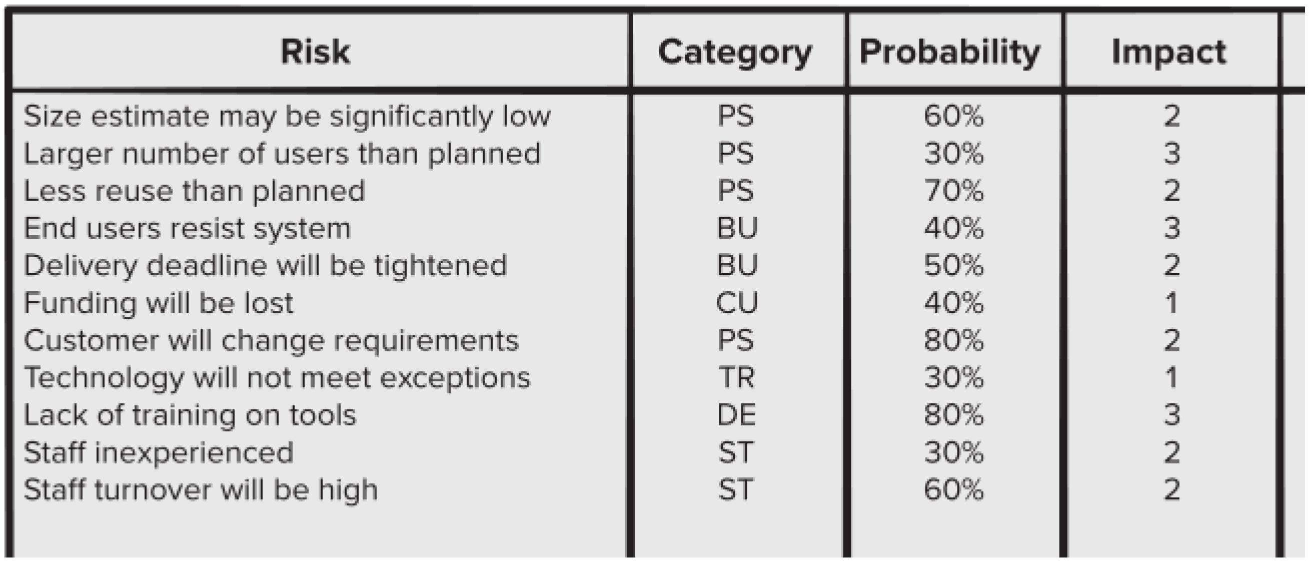
Risk Figures in Pressman & Maxim ch26
Fig 26.1 **Short summary sheet**, **partial**
o-- **Short summary** sheet (one risk per line): ID #X, **title**, **class**, **Pbb**, **Impact**, RE, Has-mitigations (Y/N)? Mitigation tasks in progress (Y/N)? – 8 columns

Missing
o-- ID#, risk index number let's you find the **long-form** (eg paragraph) **risk page** quickly

Replace "RMMM" column with these
o-- Has-mitigations (Y/N)?
o-- Mitigation tasks in progress (Y/N)?

Silly
o-- Probability too precise but inaccurate → Use T-shirt sizes instead: S,M,L (or Low, Medium, High)
o-- Impact ditto → skip 1-to-4 "new" terminology & use T-shirt sizes instead: S,M,L

FIGURE 26.1 Sample risk table prior to sorting



Risk	Category	Probability	Impact
Size estimate may be significantly low	PS	60%	2
Larger number of users than planned	PS	30%	3
Less reuse than planned	PS	70%	2
End users resist system	BU	40%	3
Delivery deadline will be tightened	BU	50%	2
Funding will be lost	CU	40%	1
Customer will change requirements	PS	80%	2
Technology will not meet exceptions	TR	30%	1
Lack of training on tools	DE	80%	3
Staff inexperienced	ST	30%	2
Staff turnover will be high	ST	60%	2

Skip Fig 26.2 3D cartesian view of RE surfaces – including Pbb, Cost, 3-rd axis = Mgmt "Concern"
o-- So, **Mgmt "Concern" is irrelevant** for RE – s/b folded into Risk Scale Class choice

Fig 26.3 **Long form Risk page** – **looks okay**, maybe too formal (rare on most med/large/XL projects)
o-- The fancy details are for contractual "work product" documents to customer (not to users)
eg, to a gov't agency

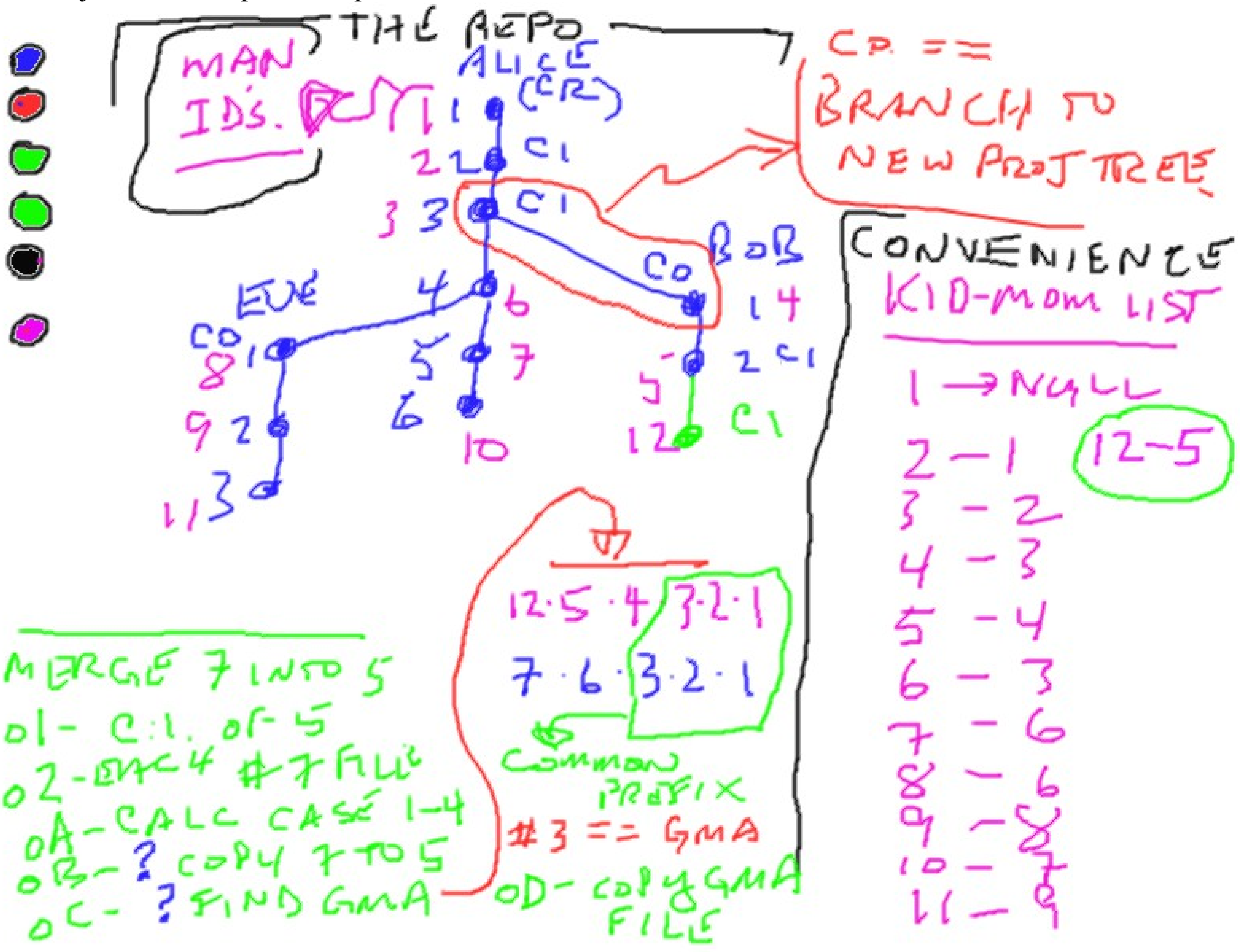
Functional Programming

o- Fcn always gives **same answer for same inputs** – like adding or multiplying
o- **Assign only once** – value to variable
  o-- Need more values, create more variables
  o-- (In compiler tech: called SSA (Static Single Assignment) – allows simpler compiler algos)
o- Use **immutable structures** – ie, create a copy instead of modifying them
  o-- Means a var always has the same value, even if it is an array or a linked list
  Caveat: An **agent** (AKA object) is typically not immutable – so think of it as an Agent

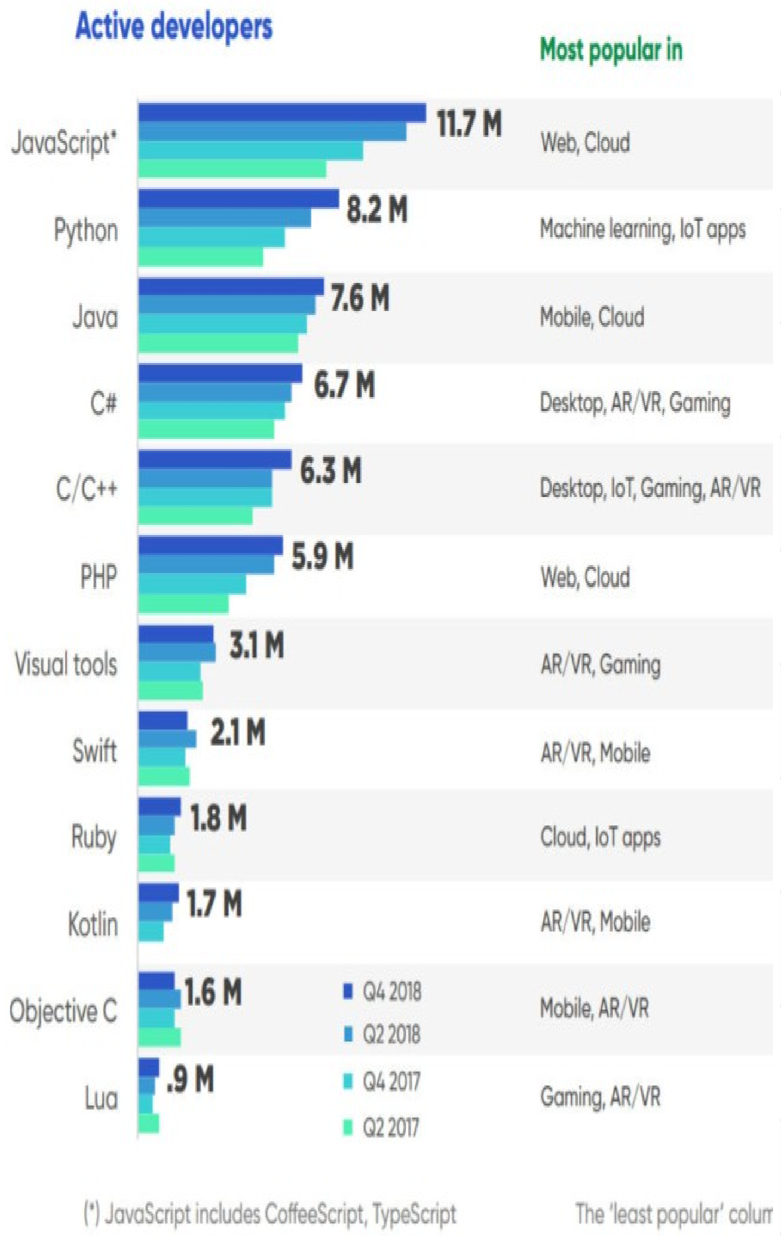
Best API is a Buffer, not a Call

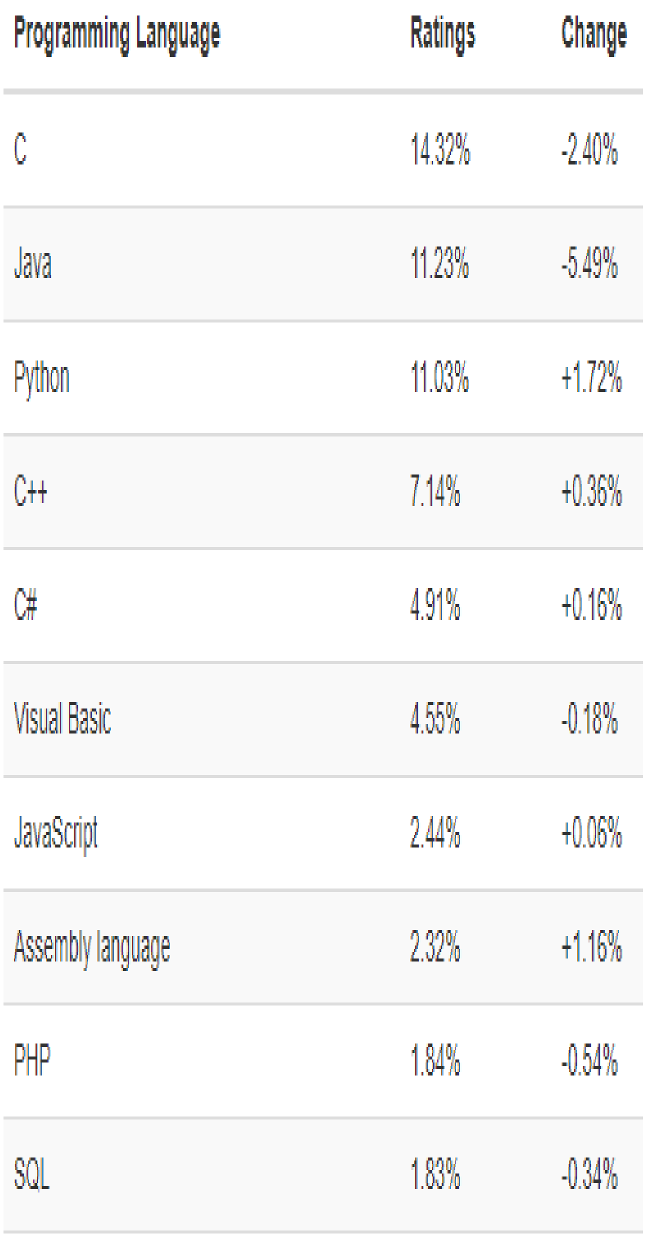
Lab

*** Project #3 due 11pm Sun April 25th



Pgmg Lang "Popularity" – Slashdata vs Tiobe "Index"





Programming Language	Ratings	Change
C	14.32%	-2.40%
Java	11.23%	-5.49%
Python	11.03%	+1.72%
C++	7.14%	+0.36%
C#	4.91%	+0.16%
Visual Basic	4.55%	-0.18%
JavaScript	2.44%	+0.06%
Assembly language	2.32%	+1.16%
PHP	1.84%	-0.54%
SQL	1.83%	-0.34%

Q: How to reduce project failures?

Idea:

-- **Brooks** (Turing) **says** Iterative Incremental Delivery wins (30yrs ago)
-- **Weinberg says** very short (**half-day**) iterations, with **demo** (54yrs ago) – Project Mercury S/W
  o- planning and writing **tests before** each micro-increment
-- **Agile says** Mini-Incremental Dev & Delivery (weeks at most)
  o- TDD = write **tests before** design/code
-- **DevOps CI says** Daily/Continuous build+regression_test of every "completed" checkin
-- **Rule #2 (Hunt) says** small Add-a-Trick & see visible success
  o- Rule #4 (EIO) says write **tests before** design/code
-- **Smart Std says** to always have **visible progress** to show, **daily**
-- **Rule #5 (Half-Day) says** plan in **visible half-day** tasks

Get the Picture? --> Stop writing large/medium chunks of untested code

Q: How to reduce program complexity? --------

Functional Programming (FP)

(Amusing side-node: John Backus (led Fortran 1957, also has 11-th Turing) invented his own FP, called 'FP')

o1- Function always gives **same answer for same inputs** – like adding or multiplying
  o-- It's FP if you can replace the calculations with a lookup table (args → answer), always the same
  Caveat: **In real life**, nearly all SW requires (**args + state→ answer**), nearly always different
    o--- state can be local, global, or both – local is simpler than global

o2- **Assign only once** – value to variable
  o-- Need more values, **create more variables**
  o-- (In compiler tech: this is called SSA (Static Single Assignment) – allows simpler compiler algos)

o3- Use **immutable structures** – ie, create a copy instead of modifying them
  o-- Means a var always has the same value, even if it is an array or a linked list or a "structure"
  o-- EG, so that you can be handed an array ref, and change slot [6], and the owner cannot see the change
  Caveat: An **agent** (AKA object) is typically not immutable (not FP) – so think of OOP as with Agents
    o--- But that's okay, cuz so are DBs and other Data-containing objects – they just add more complexity
  o-- **Immutables are Slower** to use – but We Don't Care
      → Rely on **Rule #1 (Optim)** – Be slow, until some **tiny code spot** is **Proven** to need more speed

Q: What langs are FP?

A: **Style:** All of them. You just avoid changing "in place", but make copies instead.
A: **Features:** Some default to immutable, and force you to use special "Dirty" Features for side-effects.
  o- Fast Immutable – Uses "**Persistent Data Structures**" under the hood, in the run-time mech.

Bugs Still Remaining even when you use FP:

o- **In**: Did you call it with the correct args?
o- **Mid**: Did you test it enough to be sure the Fcn calcs what you wanted?
o- **Out**: Did you get the result you were expecting for those args? A second time, too? Repeatedly?
(*) Q: But Side-Effects are what Agents's do, so **why use FP?**
  A: **To simplify 95%** of the architecture & code.

Short Fcns

o- One of the biggest benefits of writing short fcns.
  --> There is a **tendency to test** a short fcn right after it is written.
  --> This gets you closer to doing Rule #2 (No Bug Hunts), Add-a-Trick
o- Another big benefit – All the code is on the screen at one time → **eyeball** moving is faster than scrolling
  o-- plus, you don't "lose your eyeball place" while scrolling → worth 2x in productivity

More – Q: How to reduce program complexity? --------

Q: Another way to look at improving SWE success?

failure ← complexity ← info-convolving ← (#paths * their "control" variable's values → cause switching)

o- Recall **Brooks' – The Essence of Invisibility** → the architecture is too complex to "see" all at once

o-- Why important? → "see all at once" only involves eyeball movement → easier to understand it all

o-- Why even that is not enough?

→ **info-convolution** (mingling side-effects) means the arch dynamics are still too hard to follow

→ like looking at a very busy street scene and understanding what everyone is doing at the same time

o- So, How to **reduce the dynamics complexity**?

→ make the boxes/**agents simple** (and FP if possible, or some of their ops FP)

→ make each box/agent only handle **one (simple-ish) concept** (including its internal state)

o-- It can do multiple closely-related (same-concept) operations with-respect-to that data

→ try to avoid having agents talk to each other – but using buffers is more complicated

Q: What archs/patterns support more info-hiding/lower-coupling?

SWE Ease of Modding = Ease of Understanding = Reduced Complexity

o- Use SOLID/D's DinV-DinJ cutouts, to disconnect client-helper linkages

o-- Need DinJ injection mgrs

o- Use **Brokers/Mediators/.../Buffers** to disconnect direct-calls between Clients and Helpers

o-- Broker or Mediator's concept is knowing & managing **the "coupling mechanism", or "helper order"**

o-- **Buffers** can do the same kind direct-call separation but are **'dead' data**

– Only data format is known -- minimal "format" (lesser concept than an API)

– but they likely need 1+ binary flags (eg, new-data, sync flags, etc) – more complex – ? do it in FP ?

– goes under the "Reader-Writer problem"

o- Use **Entity-System IDs**, to avoid hierarchies – each entity has **multiple features**, each with a **behavior**

o-- Runtime behavioral bucket lists easily alterable

o- Use **Helper Wrappers** (eg, GoF **Strategy**), to reduce client-helper knowledge

o-- Helper wrappers hide more complex behavior

o- Use **Declarative style**, to avoid sequencing steps (AKA timing issues)

o-- Things may/must/will/need happen, but **don't care about when**

– but it's possible to create timing effects in declarative stuff – eg, with FSM state, or "current-step" var

Q: How does "virtual method" overriding work?

o Mom-Hatting – Behind the Run-time Curtain

The **Liskov** Substitution Principle (LSP) in Action -- from SOLID/D

o- Assume class Chris inherits from class Bob which inherits from class Alice. **(pic has Chris.Aleph wrong)**

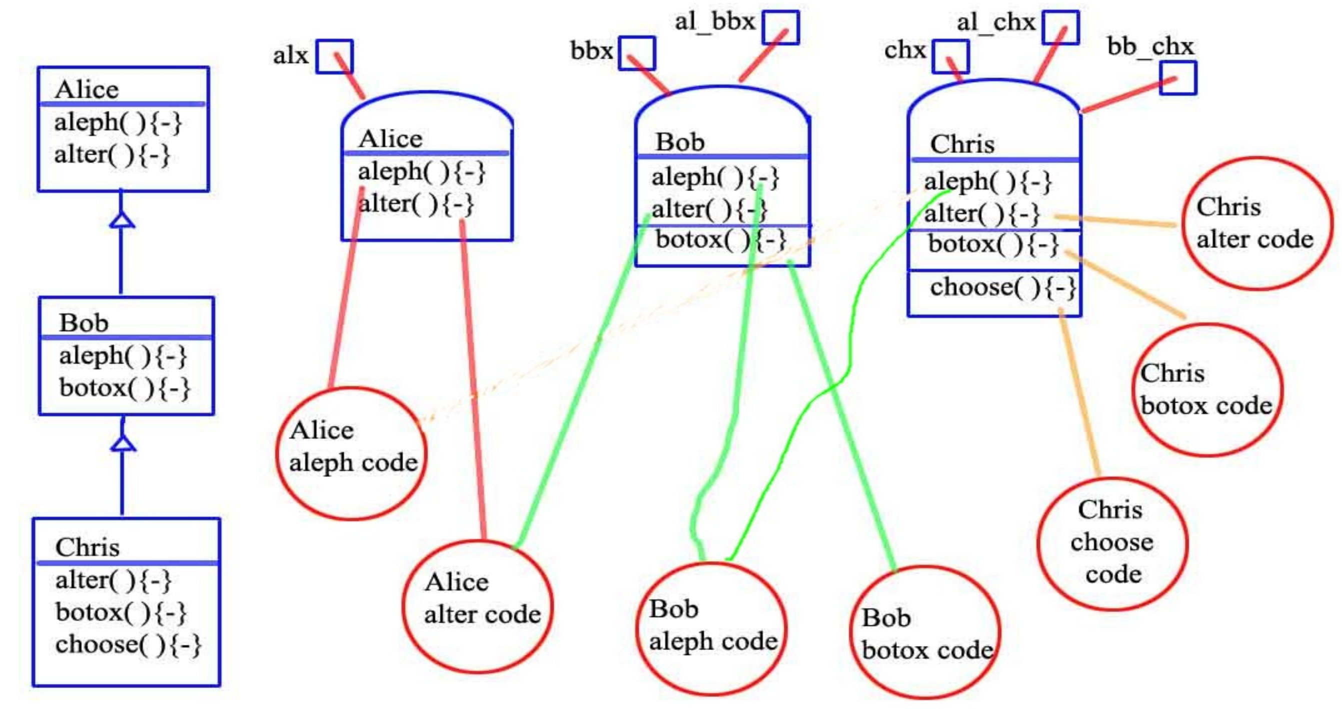
o- Class Alice defines an **aleph()** method and its kid class Bob overrides **aleph()**.

o- Alice defines a **alter()** method and its grand-kid class Chris overrides **alter()**.

o- Bob defines a **botox()** method and its kid class Chris overrides **botox()**.

o- Here are the key local object variables. o- All methods are declared public.

```
Alice alx = new Alice();        Alice al_bbx = bbx; // bbx upcast to Alice.
Bob   bbx = new Bob();          Alice al_chx = chx; // chx upcast to Alice.
Chris chx = new Chris();        Bob   bb_chx = chx; // chx upcast to Bob.
```



For each of the following method calls, **which class's method code gets called? (or a compile-time error)**

a. bbx.alter()	**Alice's**	Bob's	Chris's	Error
b. chx.alter()	Alice's	Bob's	**Chris's**	Error
c. al_bbx.alter()	**Alice's**	Bob's	Chris's	Error
d. bb_chx.alter()	Alice's	Bob's	**Chris's**	Error
e. alx.botox()	Alice's	Bob's	Chris's	**Error**
f. chx.botox()	Alice's	Bob's	**Chris's**	Error
g. al_bbx.botox()	Alice's	Bob's	Chris's	**Error**
h. bb_chx.botox()	Alice's	Bob's	**Chris's**	Error

Caveats: this doesn't use the "**Vtable**" memory-saving hack (that is always used in compiler run-times)

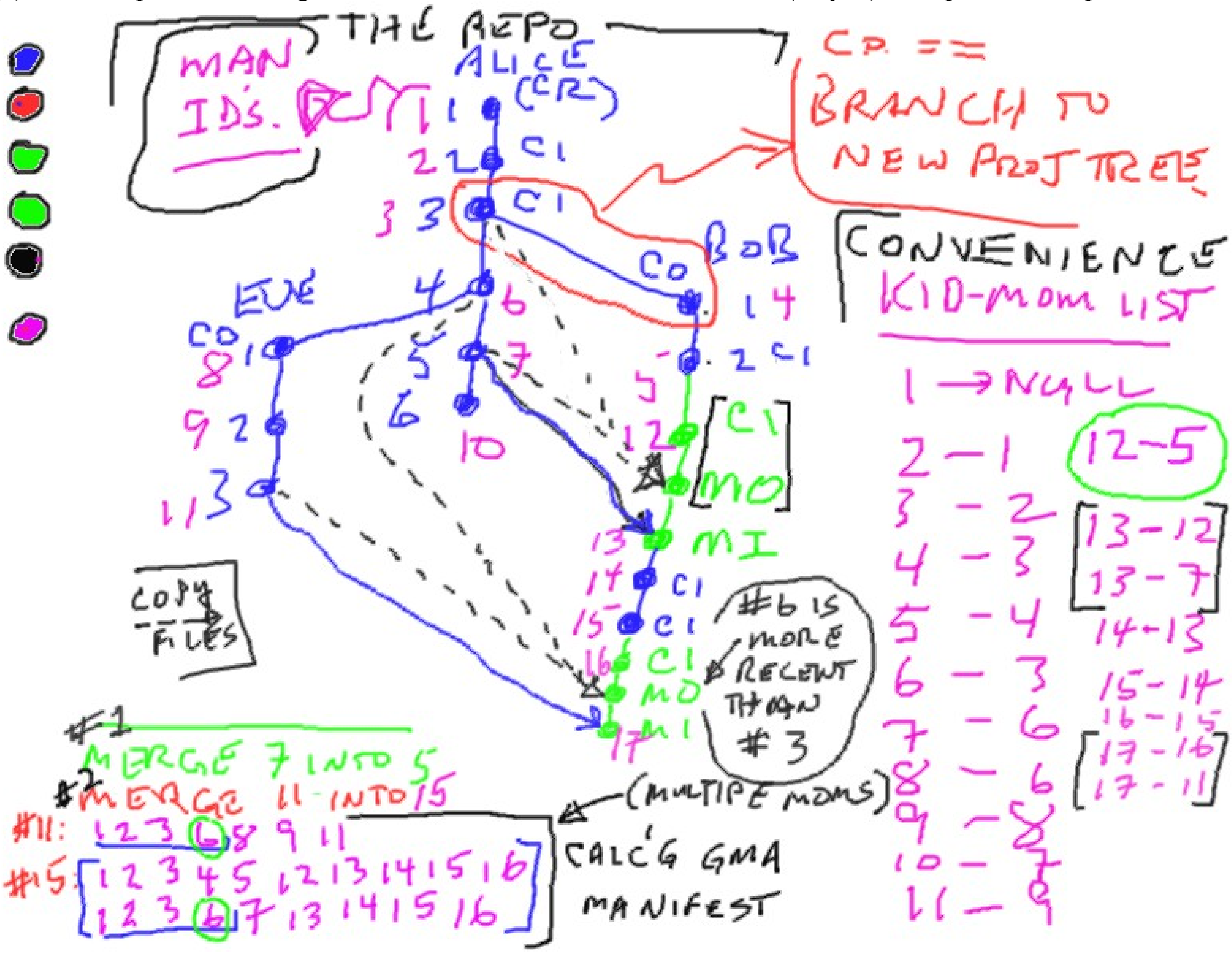
o- Also, while method refs aren't in an object (cuz vtable), the data slots ARE, and they use "class frames", too

RT memory areas: Global constants, Global vars, Code Space, RT Call Stack, RT Heap (for New/Malloc stuff)

Lab

Q: What happens when you merge? A: You get another "kid-with-two-moms" in your Repo DAG.

(*) This complicates finding the most recent common ancestor, cuz of (maybe) multiple root-kid paths.



SOLID/D – LSP – (Barbara) Liskov's Substitution Principle

(**Turing Award 2008 –** For contributions to practical and theoretical foundations of programming language and system design,
especially related to data abstraction, fault tolerance, and distributed computing)

o- "When a Kid (ref) wears a Mom-hat, it must behave like Mom."

LSP Contract Rules – But why bother with Rules? What could go wrong?

(*) Key: Don't Burn Mom's Friends

o1. Kid's override method is going to be called by Mom's Friends

o2. Mom's method can call Mom's Friends, and they expect Mom's method will work correctly for them

o-- They were calling Mom's method long before the Kid was born\

o3. Treat the Mom method as the Target and the Kid's Override Md as a before-and/or-after Decorator

For Override Methods (in Kid/Descendant Classes)

1. Preconditions cannot be strengthened

Preconditions → arg value checking

o- Mom method has restrictions on what it allows

o-- Can't Return Early just cuz Kid method thinks more processing is unneeded – cuz Mom's method didn't

o- Always – Mom method must be called -- Behave like Mom

What about Kid override method's preconditions?

Q: Can Kid method return early when Mom wouldn't?

A: No, cuz Kid method might be called by Mom/Friends who expect results (eg, to be in Mom's slots?)

(*) Key to Kid method is to **call Mom** – so Kid code can be run **before and/or after** calling Mom's method –
AND Kid method cannot mod any Mom slots

2. Post-conditions cannot be weakened

Post-conditions → return value restrictions

o- Mom method callers might rely on Mom's return value restrictions

EX: int return value is shipping cost: always positive cost

But you subclass to create Free Shipping & return 0 cost

o- Especially, return is an object of a new (sub) class – be very careful here

o- Kid wearing Mom Hat can't relax these, Mom's Friends might choke

(*) Unless: returning a subclass that does LSP correctly, too

What about Kid method's post-conditions?

Q: Can Kid method return a value Mom wouldn't (EX: ==0 when Mom returns >0)

A: No, cuz Kid method might be called by Mom's Friends, They could choke

3. During Processing – Exceptions cannot be strengthened or weakened

Q: Can Kid method throw new Exception class obj?

Q: Can Kid method handle some Mom Exception, for some new reason?

A: No to both -- not Mom behavior

4. Code Invariants must be maintained

Invariants --> stuff Mom & Friends thinks are true before & after calling Mom's (Overridden??) method

o- So, let Mom's method modify all old Mom slots & globals that Mom modifies/uses

o-- Don't modify Mom's slots from the Override method – Mom & Friends might choke on it

(***) Helpful**: Don't allow Mom slots to be directly accessed → use Mom methods instead

GRASP SWE Acronym (OOAD)

General Responsibility Assignment Software Principles – Assigning Responsibilities to Classes

General Software Principles for Assigning Responsibilities to Classes/Objects

AKA Who Does/Knows What?

[Somewhat backwards – you should Create an agent because it will be responsible for doing something
 o- Either managing a collection of related data parts
 o- Or a collection of related actions
 o- Or both
 o- And via CRC Cards, linkages, and hand-sim]

(*) Doing comes from UCs, and their helper agents

(*) Knowing is what data an Action uses/needs/creates, etc.

6 kinds of GRASP objects/agents/classes
 o- **Controller, Creator, Info Expert, Pure Fabrication, Indirection, Protected Variations**

Controller Agent/Obj

What does it do?

o- "Owns" knowledge of multiple helpers/agents

o- **Coordinates sub-tasks** and diverse objects/agents
 o-- Esp synchronization between objects/agents

o- **Gathers** resources (eg via helper) & **establishes** connections (eg via helper) to get a task done
 o-- Incl creating objects & passing them around – ie, D2=Dependency Injection (SOLID/D)
  o--- But it would use a **Creator** agent to get this done

o- **Delegates** rather than does detail work

When to use?

o- To Control/Mgr overall "system" – eg, a "Main" manager
 o-- AKA **GOF Mediator**

o- To Control/Mgr access to external device/system/component
 o-- AKA **GOF Proxy**

o- To Control/Mgr a Use Case's behavior – which is made up of 3-8 steps
 o-- AKA **GOF Mediator**

o- To Control/Mgr a Session of behavior (recall: diff between "Session" and "REST" style of doing things)
 o-- AKA **GOF Proxy** – some kind of proxy helper

Creator Agent/Obj – Building objects and handing out refs to them

Q: Who should be **responsible for creating** a class obj/agent, named B? – **Dependency Injection** (SOLID/D)

A: GoF Mediator between (helper) agent B and Callers of B (to this extent, like a Controller)
 o-- Use Cutout & Dependency Injection – to inject a ref to helper B into caller C.

A: Container/agent, named C, that agent B is a part of – but maybe use GoF Strategy, for more flexibility

A: GoF Factory pattern (build objects on request) is sometimes used for this

Info Expert – (in our terminology, an Agent)

Q: Who (in Pbm Domain) should be assigned a responsibility for some batch of specialized knowledge?
 o- Batch of specialized knowledge == Major data part – to be hidden inside its agent handler

A: Assign responsibility to Domain Expert agent with the specialized Info (about that data, and its handling)
 o- Expert may delegate to a helper agent who knows part of the problem

NB, Expert has Pbm-Domain level specialized knowledge
 Not at CompSci level – cuz that is called a "Pure-Fabrication"

Pure Fabrication Agent/Obj
o- Some stuff needs to be done, but when you look around, nobody seems to be responsible for it
o- Means you are inventing an agent/obj that doesn't exist at the Pbm-Domain level
Hence the title "**Pure Fabrication**"
Q: Responsibility & related info doesn't well-match any existing Info Expert (per the users)?
o- Some batch of coherent knowledge has no Info Expert at the Pbm-Domain level (according to the users)
o- Or, the coherent info isn't at the Pbm-Domain level, but at the CompSci level
A: Create special convenience Pure-Fab Expert agent/obj/class for it
And be on the lookout to merge the Pure-Fab class into a better already-existing class
Because it is often the case that merely creating the Pure-Fab agent will cause you to notice that what that agent handles is already almost being done by some existing agent
EX: RDB manager – ORM agent (Object-Relational Mapper – object inter-linkages to RDB tables)
A CompSci level example

Indirection Agent/Obj – we need a cutout agent (who keeps client and helper ignorant of each other)
(*) To lower coupling
Q: How to avoid Client know details about a Service Provider (AKA helper)?
A: Use Indirection – via an Intermediary and an Interface
o-- use a Cutout pattern – Dependency Inversion – with Dependency Injection
o-- **GOF** patterns **Adapter, Bridge, Facade, Observer, Mediator, Proxy, Strategy**
EX: Mediator (ORM) OO-RDB-Mapper
o-- Knows how to translate OO to RDB & back

Protected Variations Agent/Obj – involves predicting the future needs – and these are VERY expected
(*) Hide expected changes behind a wrapper so that the rest of the system need not change, too
Q: How to protect against **expected changes** to a helper agent/algo/class/subsystem?
(But first remember Rule #1 – Never Pre-optimize)
o1- as **Provider/Helper**: Wrap access to helper with a small API (hence, low coupling)
o-- and try to use simple polymorphic-friendly method signatures – if it makes sense
o-- Consider **GOF Strategy** pattern – specifically designed for this kind of thing
o- And it also allows plug-replacing agents at run-time (via Dependency Injection)
o2- as **Client/Caller**: Ensure all calls out are to Interface (mom-hatted) objs
o3- **Extreme situations**: Hide the helper behind a Controller/Mediator

Lab
Arch possibilities
Q1 How would check-in #4 know that it was checked-out from manifest #2?

Kid-to-mom file (trick)
On a manifest-creating command, create a kid-to-mom line item in a kid-to-mom file.
 o- Command has a source and a target
 Create-Repo source project-tree and a target repo
  Kid = CR manifest; Mom = null; Tree= project-tree → all info contained in CR command
 Checked-out source manifest and a target project-tree
  Kid = CO manifest; Mom = source manifest; Tree= new-project-tree → all info contained in CO command
 Checked-in source project-tree and a target repo
  Kid = CI manifest; Mom = manifest **for most recent same Tree**; Tree= project-tree
 Q: How to get it?
 A1: It would be nice if we could see that manifest in the project-tree, top folder (s/b a dot-file)
  Q: How to get the project-tree manifests into the project-tree, top folder
  A: Put a copy of the command manifest in the project-tree's top folder – for every mani-creating command
 A2: It would be nice if we could see that manifest name in a file in the project-tree, top folder
  Q: How to get the project-tree manifest names into a file in the project-tree, top folder
  A2a: Add the manifest name to the mani-list file (s/b a dot-file) for that project-tree,
    and the file resides in the project-tree, top folder
  A2b: Create/overwrite the latest-mani-name file (s/b a dot-file) for that project-tree, with that manifest name
  o- One mani-name in one std file (eg, latest-mani-name file) at top-level folder in its project-tree
checkout tree -- git branch

Recap
-- Brooks (Turing) says Iterative Incremental Delivery (IID) wins (**30yrs** ago)
-- Weinberg says very **short (half-day) iterations, with demo** (**54yrs** ago)
o- planning and writing tests before each micro-increment (= EIO)
-- **Agile says** Mini-Incremental Dev & Delivery (20yrs ago) – (25yrs for XP & Scrum)
o- TDD = write tests before design/code (= EIO)
-- DevOps CI says Daily/Continous build+regression_test of every (micro-slice) checkin
-- Rule #2 (Hunt) says small Add-a-Trick & see visible success (focus on zero run-time bugs)
o- Rule #4 (EIO) says write tests before design/code (helps focus design & avoid gold-plating)
-- Smart Person Std says to always have visible progress to show (your manager), daily
-- Rule #5 (Half-Day) says plan in visible half-day tasks
Get the Picture? --> **Stop writing large/medium/small chunks of untested code – tiny = micro-slice**
Upshot: Throw out Predictions and do day-to-day/week-to-week deliveries & hope users like it
o- This is what Agile recommends
o- Try to focus on working at a **"sustainable" pace** (non-death-march)
o- Means much less at risk at the "silver bullet" point in the project (AKA Validate the model)

Pbm
o- Make program that **works adequately** (ie, V&V is successful)
o-- Solid understanding of the **users' problem** – all important mis-communications resolved
o-- Solid understanding of how to achieve the kind of **UI the users will like**
o-- Solid understanding how to achieve the **quality the users will like**

o- Make program that that is very **easy to understand** –during new-dev and for upgrades/bug-fixes
o-- Q: Will the **As-Built docs** be available? // "As-Built" means after its built, you document it correctly
o-- Q: Will the docs be **written for newbies**?

o- Make program with group of pgmrs, **as a "team"** – cuz it's (almost) always too big for one pgmr
o-- All working together, pulling in the same direction (toward project success)

o- Make program inside the **time-frame**, and under the **budget**, available
o-- Adequate **prediction of effort**, plus adequate "contingency padding" (extra budget money)

State of the Art
o- Numerous **SWE "Quality" Groups** now exist – founded to find & tell how to "solve the SWE pbm"
o-- **SEI** founded in 1984 – **37 years** ago – spent **> $1Billion** on SWE research since then
o- Parnas/Lawford: **30+ years SWE research hasn't helped** [Parnas 2003] (ie, from1968 to 2003)
Q: Why 1968?
o-- **"SWE" coined 1968** (at NATO conf) → after a decade-plus of failed projects – big-gov big-bang
Upshot: S/W Engineering is still a black art

Pitfalls
o- **Complexity** – (Brooks has this as his focus)
o-- Scales **exponentially** – and **unavoidably** – both statically **and (worse) dynamically**
o--- And you can **NOT** test all possible static paths in the code – and we're not counting all dynamic paths
o-- Model/Arch **adequate** but **simple to understand**
o-- CS-level Arch ditto – low-coupling and high-cohesion
o- **Mgrs poor** (mostly due to lack of knowledge, preparation, tracking, and "light touch" control)

o-- Dev'r **morale** → **10x productivity** at risk (from very bad thru poor to very good)
o-- Arrange for adequate/effective **comm w users**
o-- **Looming risk** events – proactively ID, eval impact, and track risks
o-- **Gel group** into a team, and maintain the team

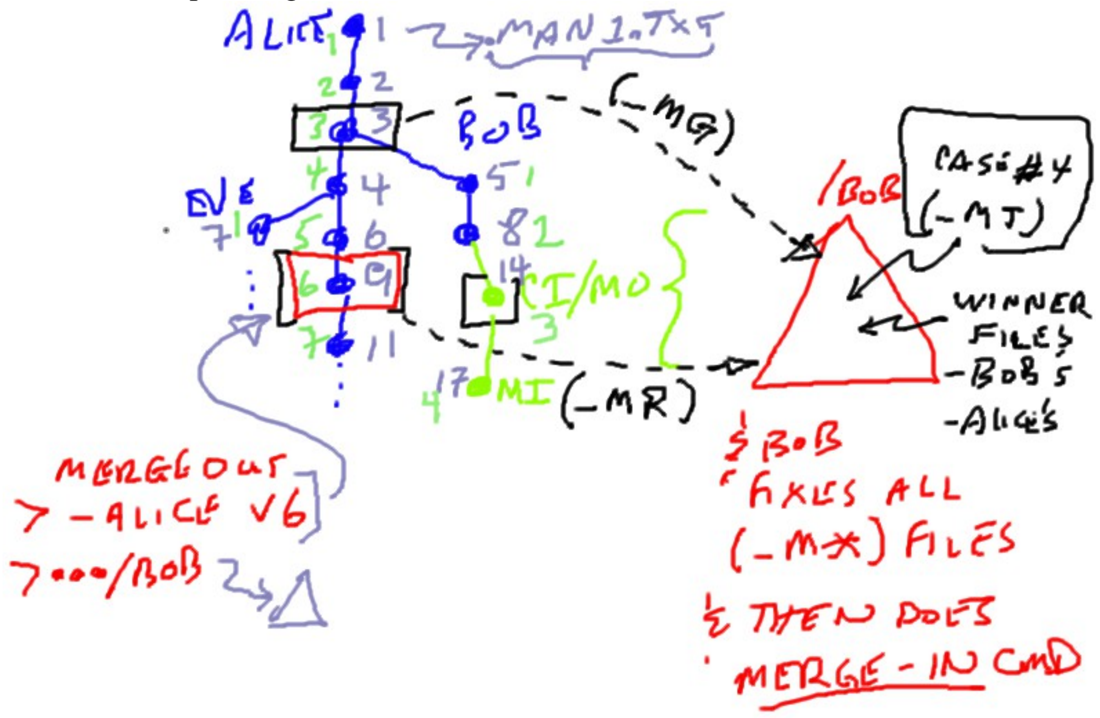
o- **Comm poor** with "users" – reqts result in building an ill-fitting, unsuitable product
o-- ID-ing who they (users) are
o-- Adequate comm time
o-- Talking in the user's (Problem Domain) language – to help uncover mis-understandings
o-- Manage their expectations to maintain their morale during comm

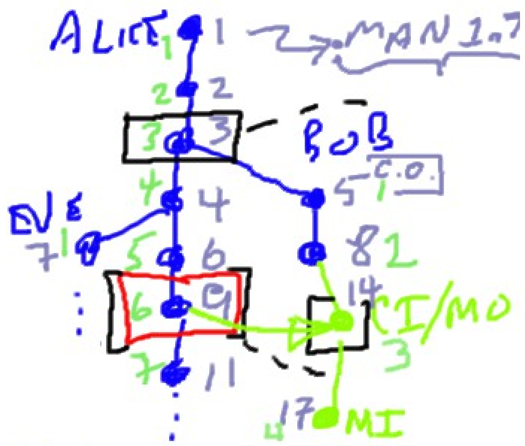
o- **Prediction Bad** of "The Plan" (Features/Effort/Timeline – AKA the "Project Triangle")
o-- "Prediction is hard, especially about the future" – Yogi Berra
o-- Manage customer expectations, avoid "bad news" surprises
o-- Avoiding "error bars" = avoiding any show of uncertainty about completion & delivery & quality

Soln: No Perfect Sol'n
o- No perfect way to avoid **Reqts changes** (users never get everything correct up front)
o- Brooks' **No Silver Bullet** – and that's just for Comm+Model correctness – **Model-phase V&V**
o- Hard to guarantee Model will have **required FURPS** (although Functionality is easiest to be confident of)
o- No perfect way to **avoid RT bugs** – so, must include "**contingency time**" to handle a "large pile" of bugs
o- No perfect way to guarantee major **integration** won't reveal major **design flaws**
o-- Large **rewrite delays are expensive** and typically introduce another "pile" of bugs
o- No way to **understand** entire program in **adequate detail** – Brooks' **Invisibility** pbm
o- No way to test all pgm interactions – the static ones are all-paths, the dynamic ones include global state
o-- Means there can be bugs lurking in untested pathways through the code

Workarounds:
Complexity –
o- Avoid building new S/W → **COTS** (where feasible)
o- Avoid major RT bugs → Rule #2 (Hunt) via **Add-a-Trick**, and build/demo in **micro-slices**
o- Avoid major integration by doing **mini-slice integration** → IID (eg, Agile style), or "**daily-build**"/CI style
o-- **One-Day Tasks** = all tasks (leading to completed testable micro-increment) take less than one day
o- **Separation of Concerns**
o-- **Separate agents** → low-coupling
o-- **SRP** in SOLID/D → high-cohesion → agent handling a concept only manages that concept, ops + data
o-- **FP** agents (as much as possible) → same inputs give same outputs (eg, like 3+4 always equals 7)
o- **Reduce complexity** in each function you build – eg, McCabe's Cyclomatic – or UC's "3-8 steps"
o-- IE, very few executable code statements (as opposed to declarative var stmts) – easy to understand
o-- Whole function fits on screen (without pan & zoom) – eyeballs are faster than pan/zoom
o--- To keep it in your head (all at once), you need to constantly refresh parts – eyeballs are fastest at this

Lab – manifest kid-to-mom parentage list – how is it created?





#1, CR /ALICE ... /REPO
#2, CI ... /ALICE ... /REPO
#3, CI ... /ALICE ... /REPO
#4, #6, #9 CI ... SAME
#5 CO MAN-3 ... /BOB
...
#14 MO MAN-9 ... /BOB
...
#17 MI ... /BOB ... /REPO

KID→MOM LIST

1 → NULL ... CR
2 → $\boxed{1}$ CI
BCUZ LAST MANI FOR ALICE IS #1.
3 → 2 CI DITTO
...
5 → $\boxed{3}$ CO MAN #3
14 → $\boxed{8}$ → LAST BOB
14 → $\boxed{9}$ → MO MAN 9

S/W State of the Art – Workarounds:
Complexity (Continued) –
o- Avoid building new S/W → **COTS** (where feasible)
o- Avoid major RT bugs → Rule #2 (Hunt) via **Add-a-Trick**, and build/demo in **micro-slices**
o- Avoid major integration by doing **mini-slice integration** → IID (eg, Agile style), or "**daily-build**"/CI style
o-- **One-Day Tasks** = all tasks (leading to completed testable micro-increment) take less than one day
o- **Separation of Concerns**
o-- **Separate agents** → low-coupling
o-- **SRP** in SOLID/D → high-cohesion → agent handling a concept only manages that concept, ops + data
o-- Functions give lower complexity than OOP classes
o-- **FP** agents (as much as possible) → same inputs give same outputs (eg, like 3+4 always equals 7)
o- **Reduce complexity** in each function you build – eg, McCabe's Cyclomatic – or UC's "3-8 steps"
o-- IE, very few executable code statements (as opposed to declarative var stmts) – easy to understand
o-- Whole function fits on screen (without pan & zoom) – eyeballs are faster than pan/zoom
o--- To keep it in your head (all at once), you need to constantly refresh parts – eyeballs are fastest at this
o- **Write code that is so simple** that jr pgmrs can understand your code
o-- NB, Functions are SRP, Objects are typically not SRP – so use Agents for big things – and lots of fcns
o-- **Avoid class hierarchies** whenever possible – they lead to Liskov bugs and OCP bugs, & low maintainab
o-- **Abstraction trap** – good useful abstractions are simple obvious proven concepts
– the original OOP point was for **reuse**, which almost always fails – requiring 3x-6x extra work to do
– so don't invent your own abstractions (as classes), but wait till you've built 3+ of em
(3+ data pts to provide solid foundation for extrapolation to a good reusable abstraction)
o-- Avoid fancy complicated OOP class stuff – fcns are much simpler – less bloat, less complexity
o-- **Don't use globals** (or more than a tiny few, if pressed)
– instead **Pass needed data as arguments** and Return any needed data changes to the caller
– and each **Agent holds a slice of global state/**data; they OWN it so only they mod it – no getters/setters
– the only globals you typically need are agents (except when starting with Rule #0, Fast)
o-- Try to **avoid passing around "object references/pointers"** – (exception, D2=dep injection)
– this changes the distributed architecture couplings dynamically, exciting (in a bad way)
– if you must, consider brokering/mediating to hide the knowledge of who currently "knows" who

Sidebar:
Single Dynamic Dispatch vs Multi-Dispatch vs Polymorphism
(Single) **Dynamic Dispatch** – pick fcn body based on 1 arg data type, the 1-st arg (to left of fcn name)
o- EX: foo.doit(...) // conceptually: call "doit" fcn with foo (ref/ptr) as the first argument
// Also, the doit body of code is specific to the foo class (if it is an override method)
o- We're selecting the correct fcn body based on the fcn's name and the 1-st argument's data type (it's class)
Multi-Dispatch – pick fcn body based on 2+ arg data types
o- EX: doit(fredx, bobx, …) // call "doit" fcn with fredx "object" + bobx "object" as 1-st two args.
// Also, the doit body of code is specific to the fred class and the bob class (& **check class hier** for body)
o- EX Alt lang syntax: fredx@bobx.doit(…)
o- EX – see GoF "Double-Dispatch" design pattern
Polymorphism – pick body for a fcn's name together with **all arg data types**
o- EX – int xx; float zz; … xx + zz; // we use the "+" fcn body that converts xx to float & does a float add
o- EX Alt syntax: add(xx, zz); // ditto
o- EX Alt syntax: +(xx, zz); // ditto, if we allow more punctuation chars to be identifier "letters"
o- To use this **args-choose-the-most-specific-body** for a fcn name (eg, "doit(-)") // check datatype hier
o- EX: print_me(…); // but in Java, there is a generic print fcn that shows an object's guts, so override it

Sidebar (Redux):

Pgm Sizes – [non-std – old-school typewritten "page" = 50-55 lines/page & 12 5-letter words/line]

XXS = 25 LOC, half page (avg typical algorithm size)

o- Max fcn size = 50 LOC, 3-8ish main lines (mostly decls, empty/brace lines, error handling)

XS = 100 LOC, 2 pgs (AKA Tiny)

SM = 500 LOC, 10 pgs – begins to be affected by poor S/W dev practices (incl mgrs)

MD = 2,500 LOC, 50 pgs

LG = 10,000 LOC, 200 pgs

XL = 50,000 LOC, 1,000 pgs

XXL = 250,000 LOC, 5,000 pgs

XXXL = 1,000,000+ LOC, 20,000 pgs

Complexity (Continued) –

o-- Use **D2 (Dependency Injection)** to provide clients with helpers, or maybe even comm via buffers

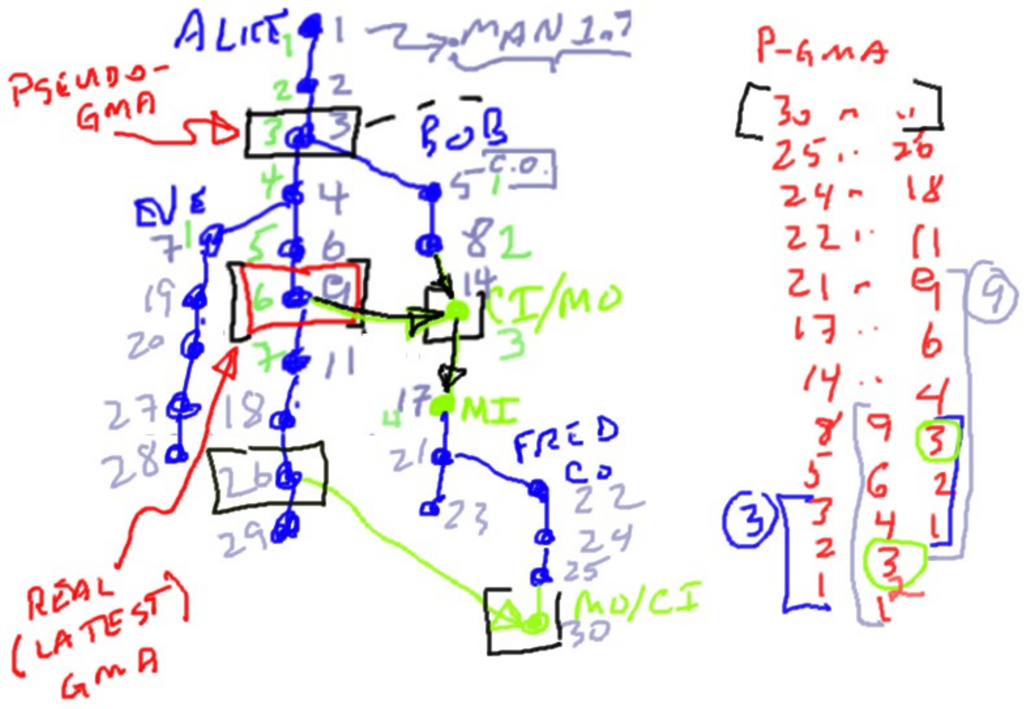
o-- **Never use getters/setters** cuz it means the object doesn't OWN its data

– instead build a structure/struct/record (AKA an object with slots but no methods of any kind)

– The reason for having "objects" is information hiding – if you're not hiding, don't use an object

Lab

Finding Grandma in a Repo with prior merges (ie, the manifest graph is a DAG, not just a Tree)



Quiz #4 Answers

. As discussed in lecture, there are three key features that are normally supported in a functional programming language. Very briefly, what are they?

A: 1) Same answer for same inputs // 4pts
2) SSA: assign value to var **only once** // lessen state confusion
3) Immutable structures: always modify a copy // outsiders unaffected

2pt = "higher level functions" // eg map reduce apply curry – produce fcns or take fcns as args
0pt = "value to variable"
0pt = "Lazy evaluation" – Don't do the work until somebody asks for it (EX, JIT = just-in-time compilation)
0pt = "Supports nested functions" – define a "local" fcn, **only callable inside its mom-fcn (scoping)**
0pt = "Efficiency" – usually FP is slower (cuz of copying before modifying a single item slot)
0pt = "Inheritance" – FP doesn't use classes, or dynamic hierarchy linkage (eg, in JS) for dynamic dispatch
0pt = "Abstraction" – doesn't do this
0pt = "Encapsulation" – doesn't do this
0pt = "Designed on the concept of mathematical functions"

map = input is a fcn & a data set X → output is a data set Y where each Y.J item = fcn(X.J)
reduce = input is a fcn & a data set X → output is fcn(X) – like add(X) → sum of X items
apply = input is a fcn & a data item Z → output is fcn(Z)
apply(fcn, Z) → fcn(Z)
curry == curry(fcn1(A, B, C)) → fcn2(B, C) where fcn2 has arg value A internally & does same thing as fcn1(-) except the A value is always used – EX: current add(3, X) → add3(X)

SWE State of the Art – Workarounds: (For Complexity, Comm, Mgrs, Prediction) (Arch)
Complexity (Continued) –
o- (More) **Write code that is so simple** that jr pgmrs can understand your code
o-- Use helper layers – lower/kids never call higher/clients –
kids are typically fcns (rarely agents), ditto grandkids and even lower layers
o-- Solve "cross-cutting concerns" (involving 2+ agents) with fcns – simpler than complicated classes
o-- **Constrain local variable scope** (where in code the var is usable) – use blocks (curly braces) to limit em
o- Write down your tests first – use EIO Rule – consider TDD tools – EIO doesn't always have to be code
o- **Build up your regression suite** while you are developing – don't break your older code that was working
o- Use the **Clean Rule** to ensure newbies can easily understand your code – and that **you understand it well**
o- Convert pseudo-code (natural lang) into **code section header comments** – preserve your pseudo-code
o-- NB, some pseudo-code makes a good section comment **but doesn't translate well** into code, eg looping
o- Make sure that everything you think is a **req't has a test(s)** (1+ EIOs) attached to it
o- Make your code **tell you what's going on** – log/assert key interim results as **proof stuff is working** inside
o- Ensure each agent does one conceptual thing (one verb) – **SRP** – **avoid noun-to-class** cuz data != class
o- Remember to **split** a task into just a **few sub-tasks** (eg, 3-8) with **reasonable conceptual names**
o-- 3-8 is for main processing path; okay to add error handling; okay to under-count for a switch
o- Solve the **main (no-mistakes) processing path first** – only then add in the error & frill handling
o-- Simplifies your fcn/method while you are making it work – much easier to understand
o- Take advantage of arch/processing explanations with **simple visuals** – post a CRC layout where all can see
o-- Use task **hierarchies**, **pole** (sequence) diagrams, **FSM** diagrams, **simple** client-helper "**assoc**" diagrams
o- Build & regression-test full system for each completed feature-slice or bug-fix
o-- Target for half-day micro-slice builds – AKA **Daily Build** (or "Continuous" Integration – "**CI**")
o-- Equivalent to Add-a-Trick for Integration – ie, integrate small pieces to **limit RT-bug locations**
o- **Learn new tech** every month – 5 years goes by fast – so does 10 years
o- Use **simple** Arch/Model **tools** – white boards, post-its, CRC cards – stuff trivial to learn & revise
o-- **Don't fight overly-complex tools** to do a simple job – "crude diagrams now" are better
o- Ensure **branch coverage** (AKA all-stmts coverage) testing
o-- If you did the EIO Rule, this should be trivial – cuz your design **only covers** the EIO samples
o- Ensure **corner-case testing** (AKA Range of Values)
o- **Agents are PAs** – they **memo** their own data – they have **more than one** closely-related action/method
o-- Each should represent **simple concept**
o-- No memoed data? **Then no agent**, just use a fcn (or a **cluster of fcns**, maybe in their own **namespace**)
o-- **Never force data to become an agent** – agent's concept w/ memoes & actions/methods is the reason
o-- A **data transformation function doesn't need to be an agent**
– fcns don't need an object instantiated to work
o- **Favor Switches** over the OOP sub-classing+polymorphism trick – switches are much simpler to understand
and the logic is all in one place, making it easy to find and modify – don't spread it out into sub-classes
o-- The OOP sub-classing+polymorphism trick **uses override methods in place of switch cases** –
and requires double the lines of code – and can lead to Liskov (LSP) bugs
– (oh, and this "polymorph" trick looks way cool, but it isn't simpler or easier to mod)
o- Use higher-level fcns in place of looping where possible
o-- map = input is a fcn & a data set X → output is a data set Y where each Y.J item = fcn(X.J)
o-- reduce = input is a fcn & a data set X → output is fcn(X) – like add(X) → sum of X items

Comm Poor –
o- Focus on reqts changes by IID (Incremental Iterative Dev) – build/delivery in mini-slices/micro-slices
o- Let users change focus after delivery of each slice – give them priority
o- CRC hand-sim with users – solves terminology & other misunderstandings
o- Use the 2-Story Rule (user pbm domain naming) to ensure users understand your architecture/model
o-- used exclusively in DDD (Domain-Driven Development) M.O.
o- Remember that the users need your help to get their PA solution – **cut them slack, be polite & nice**

Mgmt Poor –
o- Avoid most foreseeable surprises with **Risk mgmt**/tracking
o-- Brainstorm to ID events, Pbb of event (S,M,L), Cost, Impact, Mitigation/Workarounds, Tracking
o- Avoid mgmt surprises with fair (not fear) **daily standups** – ensure entire team **knows what's going on**
o-- **Be open** about your progress and immediate goals – try very hard to be transparent in your development
o-- Use the **Reasonable Person Std** – Ask for help early & show what you have
o- Ensure mgmt also does significant coding – **lead from the front** rather than from above
o- Avoid SWE def processes/tools that **invent specialty terminology** for ordinary concepts
o-- (for fun, check out RUP's phase names)
o- Remember that the dev members **need your help** to make the PA solution – cut em slack, be polite & nice
o-- Always give people a **second chance** (ie, cut them slack) – be “more than fair”
o-- Always be **open to change** – even if it means what you're working on get revised

Lab
RPC = is a function call but to a remote-CPU's function, where you wait for the function's answer
Blocking call = is a normal (or RPC-remote) function call, where you wait for the function's answer
EX:
```
  xx = foo( 42 ); // and xx "waits" for the foo(-) answer; "blocking" call
  zz = cos( ww ); // ww was set above
  yy = 3 * xx;
```
Non-Blocking call = is a function call with a “callback fcn” specified (AKA “asynchronous pgmg”)
o- where you continue without getting an answer (no wait)
o- and when the function has an answer, it calls your “callback” fcn with the answer
o- and your callback fcn installs the answer in some nice accessible variable for further use
o- and (somehow) “you are/can-be notified” that the answer is available, finally

“okay, so would an analogy for this be like ordering a pizza,
pizza shop calls you when the pizza is ready,
but in the meantime do what ever you need to do while you wait?” – James Austin Jr.
A: Yes.

SWE State of the Art – Workarounds: (For Complexity, Comm, Mgrs, Prediction) (Arch)
Prediction Bad –
o- Predict only for small time-periods (eg, Agile 2-4 weeks, or sooner)
o-- Also, exists a "**No-Estimates**" movement within Agile – instead ask them to rely on your "good will"
o- Give cust/users **confidence** with **quick repeated deliveries** of feature mini-slices
o- Give wide-but-defensible **error bars** on all estimates – maintain your personal track record as "backup"
o- Use **Half-Day Rule** for training – and to **create a personal track record**
o- Use S/M/L (**multiple) estimates** & try combining with the **discrete bell-curve eqn**
o-- Get multiple **independent** (they don't talk it over) developer estimates – to see the estimate spread
o- If you're estimating more than a staff month of effort, break it up into **micro-tasks** (half-day)
o-- Use what you know well – Half-Day Rule
o-- Remember non-work activities, allow for sleep, eat, home-life, holidays, etc.
o-- Think about sources of risk – try to estimate (S/M/L) their likelihood and impact

Mini Arch Help
o- Learn the few dozen architecture mechanisms (styles of multi-agent interactions)
o-- Others (should) know how these work – so they are easy to visualize
Buffered Arch (a data-only interface) – reduces coupling
o-- Separating two agents (or two fcns) complicates the arch, so it should be balanced with big benefits
o-- Complicates the Static Arch, but usually Simplifies the Dynamic Arch (at Run-Time (RT))
Agents – Mgrs (below) – PAs (Personal Assistants) who do/coordinate higher-level tasks
o-- Usually, this is what an "object" should be
Data Agents – GRASP "Info Experts" – wrapper for specialty data-handling, & owns the data
o-- Ie, Agents with their OWNED data – no getters/setters
Wrappers – CRC "Boundary" Agents, SOLID/D (ISP) "Interface Segregation Principle"
o-- Wrappers **provide simplified API** to Clients
o-- Can have **more than one** wrapper to the **same box** (eg, specialty Wrappers for a Data Agent)
o-- Wrapper **simplifies (AKA also reduces) access (AKA coupling)**, but **adds another box** to the arch
Mgrs – CRC "Controller" Agents, GoF Mediators etc, GRASP "Controllers"
o-- They have 2+-agent **coordination knowledge** (knowledge can also include data)
EX: they coordinate multiple sub-agents used as **procedural steps**
EX: they decide **which** sub-agents to invoke and **when**
o-- Separating (sub-) agents both simplifies arch → isolates/hides inter-agent coordination knowledge
& complicates the static arch → more boxes to understand – (so keep it simple and obvious)

Micro Arch Help – (Micro == half-day to one-day)
o- Reduce fcn complexity with
o-- Fcn should fit on screen – for better eyeball-vs-scrolling productivity
o-- Lower McCabe Cyclomatic Complexity measure – preferably < 10 for **main active exec stmts**
o--- Consider exemption for **simple Switch** multi-case stmt
– No case-with-exec-stmts that falls through
– Has a Default case – simplifies static understanding (it wasn't an MIA oversight)
– Has no branches inside a case
– Count it as 3 IFs (ie, +3 for McCabe)
o-- Use simple (newbie-ish) tech
o-- Only does 3-8-ish steps (main active executable stmts) – (maybe excepting a simple Switch)
o- Rule #4 (EIO) – written down the samples, but not necessarily as actual (say TDD) code
o- Reduce pre-test incrementally-added fcn complexity (with Rule (Hunt) Add-a-Trick)

o-- pre-test – amount of code added before compile-run-test
o-- IE, Don't add big (eg, >20 lines) batches of untested code
SOLID/D – avoid classes & especially hierarchies as much as possible
o-- to avoid OCP, Liskov bugs – and **SRP "class-blizzard"** – and D1/D2 overkill (see all, below)
DRY (Don't Repeat Yourself) Principle – means never have a code section show up in two places
o-- Two intents (points) – avoid code "bloat", and "less code means simpler (static) arch to understand"
o-- Avoid it – it is an artificial restriction that gets in the way of Rule (Fast)
GRASP –
o- Creator(Ctor) – Ctor = Constructor, Dtor = Destructor
o- Pure Fabrication – a class that is not a concept in the user's problem domain
o- Indirection(Cutout, or Wrapper) – you (or your knowledge) has to "go through" a middleman
o- Protected Variations (a Wrapper) → Future Prediction, better be very good; usually this is bad
o-- A sub-concept of Indirection
o-- Maybe use GoF Strategy, but definitely "wall off" the changable area
o-- PV isn't a class arch mech, but it is what you are protecting
FP – use it whenever you can – it's usually rarely needed – but it rarely slows down a pgm where it counts
– means **slow copying** of structured data in order to mod a slot – immutable data structures
o-- So **outsiders see no change** if they also had a ref/ptr to the structure before the mod
o-- BUT, extra CPU time rarely matters (only 1-2% of your code is a bottleneck, good odds)
o-- AND, preventing leakage to outsiders is simpler to understand (esp. under project stress)
o-- SO it **simplifies** the static arch – and especially the **dynamic arch**
– don't use it for the static/stable agents themselves
– but consider it for any data structure that is referenced/pointed-to by more than one agent
– so that one agent modifying it doesn't surprise other agents looking at it

OOP Trade-Offs

(*) Key: Avoiding** complex component-interaction **RT bugs**
o- Hard, expensive to find/fix
o- OOP has some "issues"

LSP Polymorphism vs Fancy Naming

Q: Why are we stuck with LSP?
A: Dynamic dispatch of override md based on call fcn body based on
(generic) polymorphism --> ie, **based on name + arg type(s)**, and no class hier needed
o- simple LSP polymorphism → same fcn/method name but diff object class and override of mom-class
o- Double-dispatch polymorphism → same fcn/md name but 2 diff classes, each overrides their mom-class
o- "Generic Functions" == **(generic) polymorphism**
(*) **LSP** applies to Not just OOP class hierarchy, but any polymorphism
o- But polymorphism naming makes static arch simpler
o- Hence, we trade simpler arch for possible LSP bugs
o- And single-inheritance OOP still allows LSP bugs
(*) Reason for using polymorphism – one name and (almost) any mix of arg types – & "it just works"
Alt: Fancy Naming Use common "family" name prefix with arg-types-specific suffix name:
eg, add_strings, add_string_n_int, etc.
(*) Get simpler arch but a **large pile** of type-specific **fcn names**
(altho, family+specific naming helps a lot)
o- Polymorphism was supposed to simplify this AT NO COST
o-- And its mech is "geek candy"
o- But COST is LSP bugs **if misused**

OCP vs Upgrade Class In-Place

Q: Why are we stuck with OCP?
A: Cuz we use Class Hier to get 1) Polymorphism & 2) auto Cut-n-Paste
  o- And mechs are "geek candy"
(*) But with (say) 40 old sub-classes built on mom-class,
  you need massive regression tests with very good coverage,
  Else a **mom-class mod** (which violates OCP) can cause **OCP RT bugs**
(*) But regression testing is thin for original product internals (fcns & methods)
  But good for defect fixes – AKA after-deploy bugs, not pre-deploy bugs
 o- Cuz pre-deploy tests are primarily top-level requirements tests
 o- Most unit tests, etc., are hard to include in regression suite
  ** Cuz small internal changes can ruin low-level tests – IE, they become obsolete due to dev bug fixes
   o-- And they're expensive to upgrade
   * Some TDD tries to keep up the unit (& I/T) tests
 o- Complex code-sharing can make Arch less simple, trade-off
Alt: Upgrade Class In-Place Cut-n-Paste, so no class hier needed
  o-- maybe a very shallow class hierarchy – thus avoiding **SRP "class-blizzard"**
 o- Each class is (mostly) self-sufficient – depends on no mom class
 ** Simpler Arch – (almost) one-stop to understand a class (ie, object kind)
 But during dev:
 o- Maybe mistakes using cut-n-paste to build similar classes
  o-- EG, mod one class & cut-n-paste mod to a dozen others
  o-- Esp, mod one method & cut-n-paste (slightly modded) to a dozen others
 o- Code & memory code "bloat" -- duplicated code
  o-- Memory exe size ("memory footprint") a big pbm in "old days"
  o-- Mistakes propagating class method mods cause RT bugs
  o-- Each class md likely has tiny class-specific stuff, but the bulk is code copied verbatim
 o- Pgmr "sneak-around" trying to share code in weird ways – reinventing RT class supt
  o-- NB, these weird ways are usually class-hier built-in stuff hidden in the RT supt
    EG, "rolling" your own VTables, class "frames"

SRP vs Lemmas

  o- SRP (Single Responsibility Principle)
Q: Can SRP be bad?
A: Collecting and en-classing ALL tiny bits of commonality
 o- Each in its own class in its proper spot in the hier
 AKA "Factoring out" commonality – a useful math-ish technique – NB, this is unrelated to "Refactoring"
 o- Jockeying the class hier as each common piece is found
(*)* Leads to "**Blizzard Of Classes**" – **SRP "class-blizzard"**
 o- Often under pressure to abide by DRY (Don't Repeat Yourself)
 o- Complicates arch, under guise of (math) "factoring"
 ** But math-ish mechs are "geek candy"

(*) Most situations are not complex Trees (kid = sub-class) – at the problem domain level
  o- where you are converting nouns to classes in a class (tree) hierarchy
 But rather, most are complex DAGs (common kid & many moms) – (and some are general digraphs)
 o- A single object can have multiple attributes → means object belongs to multiple clusters/groups
 (*)** Very common arch fail is using trees that s/b DAGs or a list of groups each object belongs to
  o-- Cuz, it's easy to see the tree relationships at first
  o-- And modding a big Tree is very hard
    due to very strong hier coupling (CF OCP) – and it is worse when faced with an **SRP "class-blizzard"**
    (although compile-time errors help a bit)

Alt: Lemmas Reserve SRP for **known named concepts** (as it should be)
 o- Math actually does this:
  o-- "Theorem" is (close to) a named concept
  o-- "Lemma" is a wrapping up of **convenient commonality** – ("husk" in Latin)
 (*) Don't force convenient commonality to be a class,
  Reserve classes for well-named concepts (eg, in Pbm Domain)
  o-- "Lemmas" are GRASP "Pure Fabrications", and s/b rare (or well-known comp sci things, eg Hash)

SOLID's D1/D2 vs DIY Helpers

D1/D2 = (Dep-Inversion)/(Dep-Injection)

Q: Can D1/D2 be bad?

A: Maybe

Recap:

D1 (Dep-Inversion) -- use abstract/interface cutout class
  o- So, changes to Helper guts (but no change to API)
    don't cause recompilation of Client
  (*) But, in most OOP Langs, using Ctor causes this recompilation (even if using a cutout interface class)
      o-- cuz using the Helper's Ctor requires knowing ("including the header") the Helper class, too
    So, still need to use D2 – meaning you don't do the Ctor to create the Helper agent

D2 (Dep-Injection) -- special Helper Builder **injects** (links up) Client with Helper-ref at run-time
  o-- Client uses cutout class also (D1)
  o-- Guarantees non-API mods to helper (class) guts don't recompile Client
 ** And D1/D2 mechs are "geek candy"

Alt: Client Builds/Owns Helper **till proven Need**
(*) For most situations Helper only works for Client
  So, Client building Helper isn't a problem, cuz Client OWNS Helper
  AND, DIY is a simpler arch
** But, slightly inhibits generalizing/"factoring out" Helper as a reusable part (Caveat, Rule #1)
    o-- "**Factoring out**" as "splitting out & making it its own thing (fcn, md)" in the mathematical sense
  o- Cuz Client-specific Helper API likely needs modified,
    Meaning Client will need to be modified,
    OR Client won't use the new Helper (so some duplicated code)
  (*) Avoid complicating arch with extra one-off cutout class – (**trades simpler arch for lower-coupling**)
   o- And DIY avoids Pre-Factoring Before Need
    (*)* Pre-Factoring Before Need violates both JIT and YAGNI
     JIT = Just in Time -- wait till proven need
     YAGNI = You Ain't Gonna Need It -- almost always

Alt #2: Client calls Mediator/Middleman for help, then Mediator calls Helper
  o- Adds a Mediator/Middleman agent into the mix
  (*) Avoid Client knowing anything about Helper – Mediator can even transform the API
  o- Also, this is **doable in non-OOP styles**

Helper Class vs Single Fcn

Q: Can a Helper Class be bad?
A: If it's only a single method, then
INSTEAD of calling a named fcn? – AKA "Procedural Style Coding", which is usually simpler code
Why add extra code to
o1- **Define** a Helper class, and
o2- **Instantiate** a Helper object, and
o3- **Pass** around a Helper object ref/ptr, and
** And do you need an Extra class to Construct the Helper obj, so you can do Dep Injection?
o4- Have to **call** for help **via** the Helper **object ref/ptr**
(*) This adds **extra Code bloat** and (slightly) complicates the arch
Q: How about a Helper with 2 methods?
A: How related are those 2 methods?
o- Couldn't they be just 2 named fcns?
o- Is this a misplaced attempt at grouping?

Quality Assurance (QA) – Assuring likelihood of project success w.r.t. Validation (ie, users like result)
o-- First, of course, assure **project gets to deployment phase**
o-- **Second** – make sure **process** isn't getting in the way of (development) **progress**
– Q: How to know? A: Fewer mtgs (on subjects below), & more progress on Features completed
o-- NB, QA is a Mgr issue, but it may be imposed on the project team (incl mgr) from higher-up in biz

Ensure --
o- **VCS/CMS** for all notes, EIOs, docs, code, tests, logs – find, or rollback to, any version of any item
– NB, non-code docs & notes rapidly go out-of-date as system is being built
o- **Users** (more broadly, stakeholders) **approve** of features/reqts & product quality levels (ilities)
– to the best of their ability (cuz users aren't compsci tech knowledgeable)
o- **Bug Tracking** process for all bugs found in code that has been submitted to VCS as done/tested
– NB, sometimes low-level bugs may be left in place to meet delivery schedule
– But users should be apprised of these
o- **Reqts EIOs** before/during UC development – so that EIOs drive UC design – every reqt s/have a test(s)
– (NB, **EIOs** at all levels **should drive design** at all levels)
o- **Users agree with EIOs and UCs before**/during UC & arch/model/CRC development
o- **Users agree with preliminary arch/model** – eg, via CRC hand-sim, or fancier prototypes
o- **Detailed/Unit EIOs before**/during Detailed Design & Unit Design – less gold-plating or future reuse
o- **EIOs for testing** of component and unit **APIs** – so devrs can do pre-integration API verification
– reduces integration bugs (which can still happen – **APIs** merely help, but **aren't the entire coupling**)
o- **Unit testing** (with EIO data) **is performed** – & test "scaffolding" with EIOs & run logs go into VCS
o- **Component-level integration testing** (with EIO data) is performed – ditto
o- **Reqts-level testing** (with EIO data) is performed – ditto
– & reqts tests go into **Regression suite**
– NB, lower-level tests typically require scaffolding code, which complicates use in Regression suite
o- **System internals description docs** are revised to reflect actual product (ie, "**as-built**" docs)
o- **Major bug**, causing major system (eg, including some arch) changes, should also **include EIOs**, etc.
o- When behind schedule, **devrs re-estimate** their tasks – as often as weekly
o- When behind schedule, **users** are allowed to **help revise project reqts/features priorities**

SEI CMM (CMMI) – Capability Maturity Model (, Integrated)

o- For **Evaluating likely success of a business** running a S/W project – rate a business selling "fresh S/W"
  – Success = on-time, under budget, features desired, validation by users

o- Classifies software dev organizations into **5 levels**:

Level 1 : Initial (eg, "free-for-all")

o- These organizations tend to use **code-and-fix** style development.

o- Software development has no written (auditable) process – anarchy.

o- **Projects tend to run over** budget and behind schedule.

o- Organizational knowledge is contained only in the minds of individual programmers;
  SO, when a programmer leaves an organization, so does the knowledge.

o- Success **depends largely** on the contributions of individual "**hero**" programmers (ie, "strong" pgmrs).

Level 2 : Repeatable

o- Basic project management **practices** (**written**) are established on a **project-by-project** basis,
  and the **organization ensures** that they are followed – Q: How? A: via QA auditing.

o- Project **success no longer depends solely** on specific individuals – (ie, devrs are "plug-replaceable").

o- The strength of an organization at this level depends on its **repeated experience with similar projects**.

o- The organization **may falter** when **faced with new** tools, methods, or kinds of software projects.

Level 3 : Defined – (AKA you build it well, and you use written processes for all projects)

o- The software organization **adopts standardized technical and management processes across the org**,
  and individual projects (within the org) tailor (in writing) the standard process to their specific needs.

o- A **group** within the organization is **assigned responsibility** for software **process activities**. (eg, QA)

o- The organization establishes a **training program to ensure** that managers and technical staff
  have **appropriate knowledge and skills** to work at this level.

o- These organizations have moved **well beyond code-and-fix** development,
  and they **routinely** (but no percentage indicated) **deliver software on time and within budget**.

Level 4 : Managed – (AKA you collect blizzard of "useful" data on all projects, ideally for improvement)

o- Project outcomes are **highly predictable**.

o- The process is stable enough that **causes of variation** (in outcomes) can be **identified and addressed**.

o- The organization **collects project data** in an **organization-wide database**
  to evaluate the effectiveness of different processes.

o- All projects follow organization-wide **process measurement standards** so that the data they produce
  can be meaningfully analyzed and compared.

Level 5 : Optimizing – (AKA you have a meta-process to improve your processes, & hence success rate)

o- The focus of the whole organization is on continuous, proactive identification and dissemination
  of **process improvements**.

o- The organization varies its processes,
  measures the results of the variations,
  and diffuses beneficial variations as new standards.

o- The organization's quality assurance (QA) focus is on defect prevention
  |through identification and elimination of root causes.

Caveats :

o- YMMV – AKA "gaming the system"

o-- "**Potemkin**" (AKA fake) **processes** in order to gain CMM cert – but SEI samples employees on practices

o-- **Buying "influence"** with accreditation organizations

o-- **Hiring senior accreditation mgrs** into your business after positive Level certification – quid pro quo

o- Why? There is tremendous money available to companies with CMM Level certifications

o-- Can a biz maintain such a facade in the long term? – Maybe? – Best way is to do the Mature thing well

Run-Time Environment – Compiler-based Languages

RT Memory Areas – 5 Kinds (for most major langs)
o- **Global Constants** – that can't fit into a machine instruction – usually > 8- or 16-bit (instruction upper limit)
o-- EX: string constants, like "Hello there" – EX: Vtables for the classes
o- **Global Variables** – accessible from all code, (except when global names are shadowed by local names)
o-- Includes "**private globals**", eg C/C++ keyword "auto" added to a local var makes it a private global
– Local → only accessible from inside its fcn or block
– But in Global Vars, not in the fcn's local Call Frame on the Call Stack
– Hence, var value survives fcn return and is still there on next call to that same fcn
– AND not accessible from outside its scope (AKA inside its fcn or block, or block-remainder)
o- **Code Space** – both global fcn code and nested (ie, local, hence private) fcn code is here
o-- **For regular fcns**, caller has exact address of code – uses a "call" or "jump-to-subroutine" instruction
o--- Compiler sets a symtab (symbol table) index for the fcn name/symbol in the "call" instruction
– AND the Linker (which lays out Mem Areas) replaces the index with the fcn's address
o-- **For dynamic dispatch fcns**, caller ends up jumping thru offset in some **VTable** (a Constant table)
o--- Compiler sets up an indirect call through the named VTable and with an offset to the fcn/md name
– AND the Linker replaces the VTable symtab index with the VTable's address in Global Constants
o- **Call Stack** – (invented to handle recursion, and later all CPUs added H/W support for it – SP = Stack Ptr
o-- **Every fcn/md call** creates **Call Frame**, pushed onto the Call Stack
o--- **Call Frame** – has 1 slot for "return address"
for (next instruction to run, caller's next, or its mom's next – specific to CPU+Lang_Runtime)
– has "parameter" slots (local var slots) for all its arguments
– has enough slots to store all its local vars (those active at the same time, so <= the actual #vars)
o--- Allows multiple copies of a recursive fcn's parms & local vars to exist at the same time (easily)
– Before this, you built & manages your own stack in code to do recursion
o--- Earliest major lang to supt recursion – as with most things, Lisp 1958
o--- Likely still some extreme embedded pgmg systems that "can't afford" recursion
o- **The Heap** – not a "heap tree", but just a large area ("pile") of bits – leftover
o-- All dynamic data is placed here – a data-sized pile of bits, a "**block**", (with random values) **is "allocated"**
o-- When the pgm is done with a dynamic object/structure/array, the **memory is "reclaimed"**
o-- The **Heap Mgr** (part of the lang's RT mech) handles **allocating and "reclaiming"** memory blocks
o--- It usually has a **"Free List" of unallocated blocks**
– at startup, one really big block (usually)
– **after** running awhile (ie, **allocating and reclaiming**), it has a bunch of disconnected blocks
– AND between disconnected blocks are allocated "holes"
– AND the **Free List** is typically a **linked list of free** (currently unallocated) **blocks**
o-- **Reclaiming** is done either by the Pgm/Devr **calling a "free-me" fcn** → calls the Heap Mgr
– OR **automatically** done by the "**Garbage Collector**" (AKA "**GC**")– present in some langs
o-- The "Garbage Collector" does fancy reclaiming – (1-st was Lisp, of course)
o--- Like recursion, GC was once thought to be too expensive (slow and plump)
o--- Fast GC algos (for Lisp) were invented in mid-80's – but any lang can do this
o--- Interpreted langs (which are usually about 30x slower) typically don't worry about GC speed
o-- Without GC, when no single block is big enough for an alloc request, the Mgr tries to merge blocks
o--- The Free List can end up with side-by-side blocks because it is (thot) too costly to merge on return
– Cuz the Free List isn't kept in block-address order (costly to resort List on every block return)
– So the Heap Mgr only tries to merge on "can't find block big enough"
o-- Either way (**GC or not**), if big enough block isn't available, pgm might crash (or throw exception, etc.)

(in Facts and Fallacies of Software Engineering, by Bob Glass, 2003)

Fact 2
 o- The **best** programmers are up to **28 times better** than the worst programmers,
  according to “individual differences” research.
  Given that their pay is never commensurate, they are the biggest bargains in the software field.
 Caveats (it's very controversial):
  o-- Many **senior pgmrs disagree** both with this kind of “fact” and with some of the research studies that initially reported this kind of result (eg, 10x or more difference). Including at major biz (eg, Google).
  o-- OTOH, there are **many anecdotes** of **extremely strong pgmrs** who exhibit this difference.
  o-- OTOOH, it is often claimed that such extremely strong pgmrs are “prima donnas” that **cause social pbms**.
  But, that kind of defeats the claims that there are no such pgmrs.
  o-- OTOOOH, the 80-20 Rule (for pgmrs doing the work) means 80% of the S/W coding
  is done by 20% of the developers – 80-20 Rule is Pareto's Rule, Pareto stat distribution of who/what contributes the most, least.
  → in 343, the “80-20 Rule” means 80% of the bugs come from 20% of the code
  Pareto Distribution is followed more accurately in larger systems (where differences are “washed out”)

Fact 29
 o- Programmers shift from design to coding
  when the problem is decomposed to a level of “primitives”
  that the designer has mastered.

  If the coder is not the same person as the designer,
  the designer’s primitives are unlikely to match the coder’s primitives,
  and **trouble** will result.

Fact 33
 o- Even if 100 percent test [branch] coverage were possible [in a large pgm],
  that is not a sufficient criterion for testing.
  Roughly **35 percent** of software **defects** emerge from **missing logic paths**[branches] ,
  and another **40 percent** from the execution of a unique combination of logic [branch] paths.
  They will not be caught by 100 percent [branch] coverage.
– Nox below here
Fact 41
 o- Maintenance typically consumes 40 to 80 percent (average, 60 percent) of software costs.
  Therefore, it is probably the most important life cycle phase of software.
Fact 42
 o- Enhancement is responsible for roughly 60 percent of software maintenance costs.
  Error correction is roughly 17 percent.
  Therefore, software maintenance is largely about adding new capability to old software, not fixing it.
Fact 44
 o- In examining the tasks of software development versus software maintenance,
  most of the tasks are the same
  — except for the additional maintenance task of “understanding the existing product.”
  This task consumes roughly 30 percent of the total maintenance time
  and is the dominant maintenance activity.
  Thus it is possible to claim that maintenance is a more difficult task than development.