planning

top-10 recap

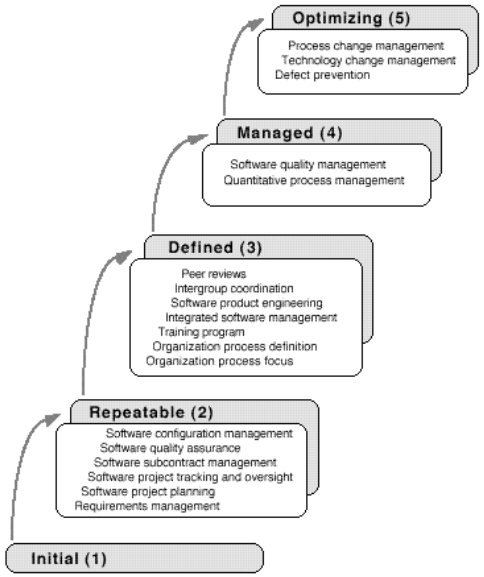
- discussed the top-10 essential practices for software development:
  1. source code control
  2. issue tracking
  3. build automation
  4. automated regression tests
  5. **release planning**
  6. design specifications
  7. architecture control
  8. effort tracking
  9. process control
  10. business planning

capability maturity model

- classifies an organization's maturity into 5 levels
  - each level prescribes a group of practices
  - CMM is also a road to process improvement
  - must have all lower-level practices in place before attempting next level
- can be certified to a certain CMM level
  - some similarities to ISO 9000
  - not universally agreed to be a good thing, but is an interesting exercise

capability maturity model (2)

capability maturity model (3)



The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

relationship to ISO 9000

- ISO 9000 is a set of quality standards
  - subset of these are specific to software
  - must document the process
  - must maintain "quality records"
    - used in audits to ensure adherence to the process
  - process can be anything

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

relationship to top-10

- top-10 practices are necessary to achieve CMM level 2 (repeatable)
- also, top-10 includes enough level 3 (defined) stuff to attain ISO 9000 certification
- and, top-10 even includes some level-4 (quantitatively managed) stuff, where most useful
  - defect arrival/departure rates
  - estimate vs. actuals

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

planning

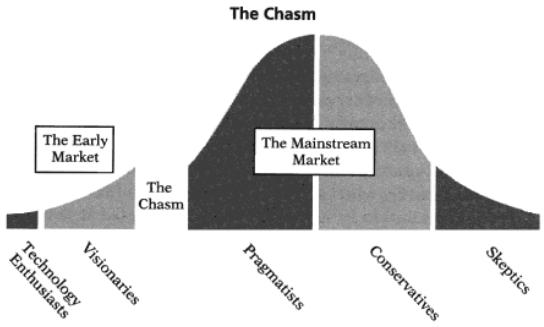
- planning is the most important aspect of CMM Level 2
- common flaws regarding planning
  - making no plans!
  - make a plan, but don't track it
  - attempt to track the plan with inadequate tools
    - Gantt charts
    - Microsoft Project

why plan?

- planning isn't *always* a good thing
  - release/expected date is not important
  - no expectations on new functionality
  - proof-of-concept (a.k.a. "spike")
- planning is required when external pressures come to bear on feature availability dates
- doesn't usually apply to first releases, but is necessary to "cross the chasm"

crossing the chasm

- book by Geoffrey Moore (1991)



planning essentials

What are we building?
By when will it be ready?
How many people do we have?

- answer these questions, and nothing more
  - not "who will be doing what?"
  - not "what are the detailed tasks required?"
  - not "in what order must the tasks be performed?"

implementation plans

- once initial planning is complete we can transition to a more detailed development plan
- this more detailed plan sorts out:
  - who is assigned to what
  - dependencies between features
  - etc.

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

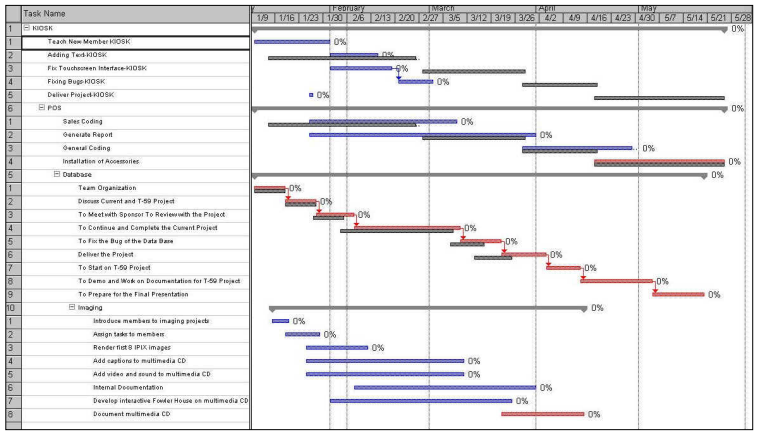
of mice and men

"The best-laid schemes o' mice an' men Gang aft agley"
– Robbie Burns

- the essence of planning is uncertainty
  - plans never go "according to plan"
  - must embrace change rather than resisting it
- how to make plans **and** embrace change?
  - track the plan constantly, not just at the start
  - react quickly & decisively to adverse situations
  - embrace a change in direction
  - re-plan quickly, can't be hard to deal with unexpected changes

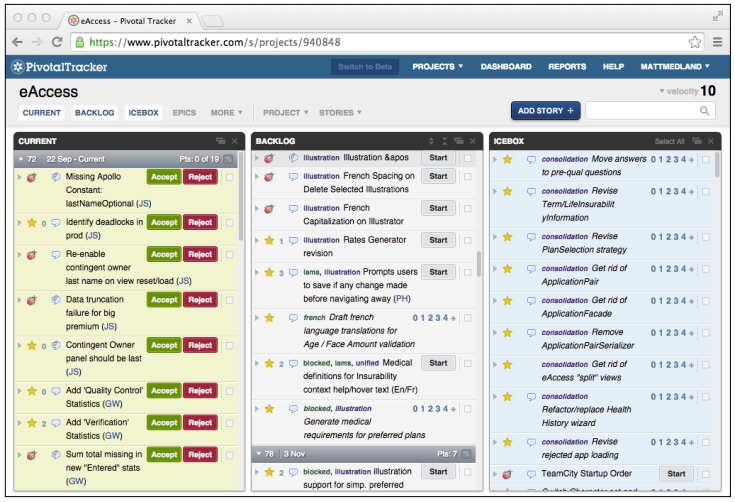
The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Gantt charts == harmful



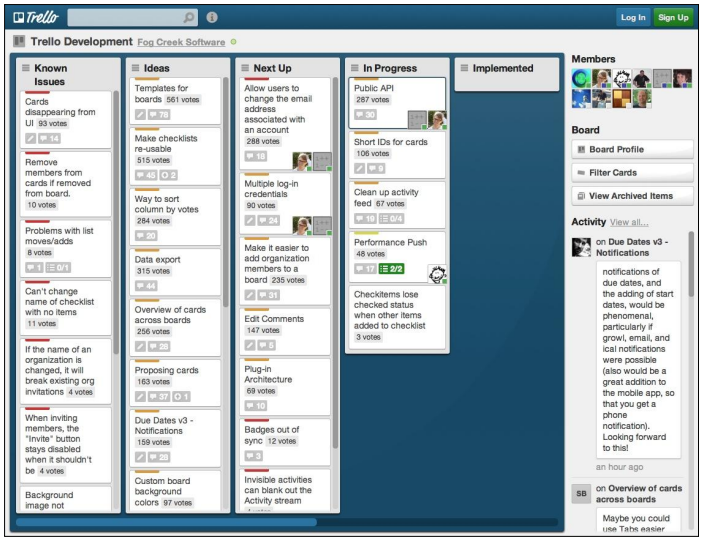
The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

agile planning with pivotal



The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

storyboarding with trello



internal changes

- estimation errors
  - initial estimates contain a significant (usually one-sided) margin of error
  - as plan progresses, and more information becomes available, variance in errors drops
- developer availability changes
  - illness, parental leave, resignations, cut backs, unexpected vacation plans, unexpectedly low hours of work, unexpected low productivity

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

external changes

- new (big) customer with specific demands
- pressure from competition
- collaboration opportunities
- acquisitions & mergers
- sudden changes in customer needs
  - ex. regulatory changes that affect them

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

the difficult question

- what are we building?
  - hard for 1st release, later ones have big wish list
  - marketing/product manager pick ones that will get most sales
- by when will it be ready?
  - too soon: customers won't be ready, won't want to learn, install, pay for it
  - too late: competition will pass you, customers will forget you == forgone revenue
- how many developers?
  - usually fixed for a given release, or planning horizon

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

the difficult question (2)

What are we building?
By when will it be ready?
How many people do we have?

the difficult question is:

can we do all 3 at once?

a common problem

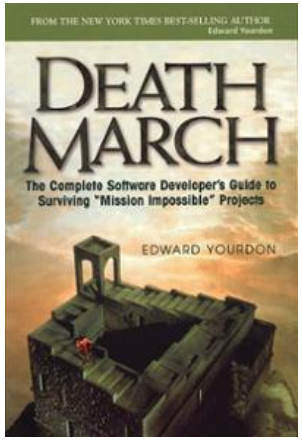
- often organizations will answer all 3 questions, but not address the difficult one
- development mgmt. wants to please the rest of the company and agrees to too much – gung-ho spirit!
  - some actually believe in over-commitment to boost productivity – "it's a stretch, but we'll pull it off!"
- developers will say "it can't be done!" – but that's all those folks ever say, right?

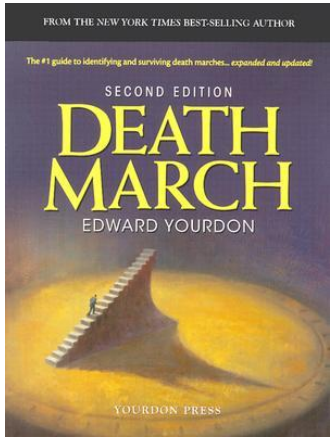
a common problem (2)

- major state of denial sets in…
  - or sometimes hopeless optimism
  - everybody is secretly hoping for a miracle
- nobody will accept any blame, and why should they?
  - dev. mgmt.: "we told you it was a stretch!"
  - developers: "we said it couldn't be done!"
  - marketing & sales: "R&D, should have said something earlier!"
  - CEO: "you all told me everything was fine!"
  - Yourdon's death march…

a common problem (3)

- *Death March* – Edward Yourdon





the solution – good planning

- the "*death march*" doesn't need to happen
- to avoid it we need some courage and conviction
- also need common sense:
  - is it even feasible to do what's asked by the date required?
  - don't give a quick (off-the-cuff) answer even if it's obviously impossible
  - put together a plan to demonstrate the facts.

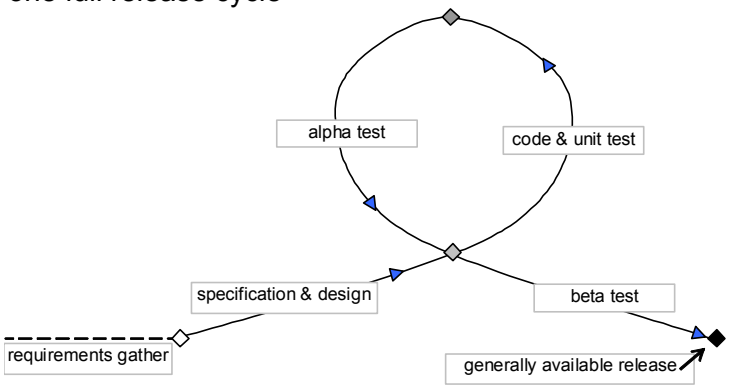
agile horizon planning

- provide a software planning framework
  - that balances
    - business concerns
    - software development concerns
  - provides better predictability of
    - end-date
    - delivered defect-minimized feature set
  - provides early notification of slips
  - allows for re-planning as events unfold
  - deals explicitly with uncertainty

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

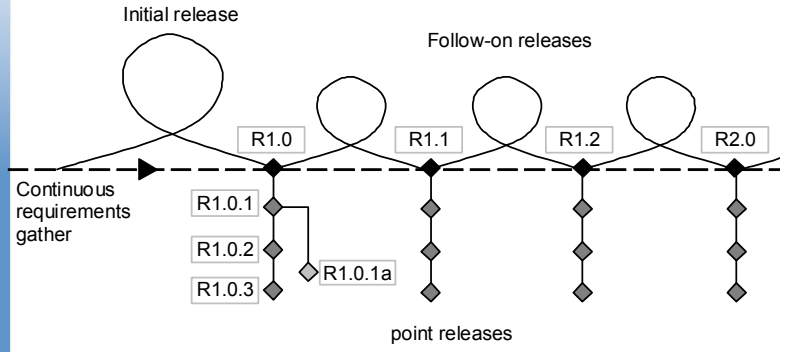
product lifecycle

planning horizon is
one full release cycle



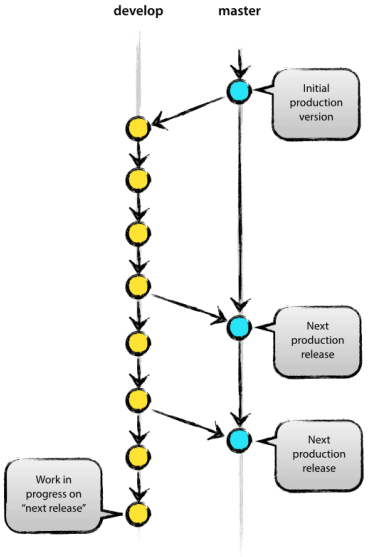
The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

follow-on lifecycles



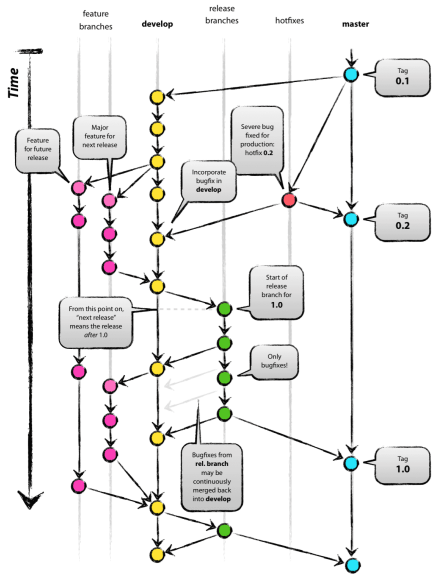
The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

simple Git branching model



The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

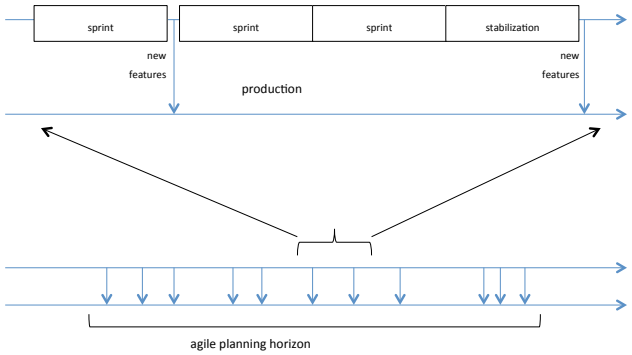
complex Git branching model



The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

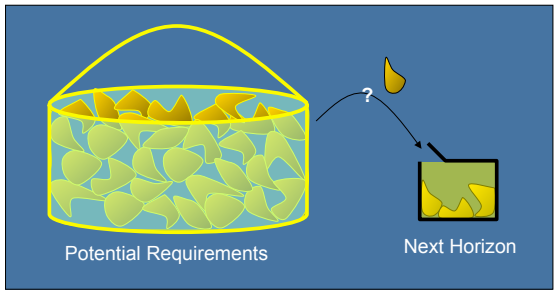
SaaS lifecycle

- more frequent release of code to production
  - forced upgrade – spreads the risk
  - low release overhead – possible
- planning horizon is according to business convenience or planning necessities



The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

eliciting potential requirements



- starts with a wish-list
- stated as business requirements
  - features for architectural enhancements

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

simple release plan

Dates: Coding phase: Jul.1—Oct.1
Beta availability: Nov.1
General availability: Dec.1

Capacity:	*days available*
Fred	**31** ecd
Lorna	**33** ecd
…	…
Bill	**21** ecd
total	***317 ecd***

Requirement:	*days required*
AR report	**14** ecd
Dialog re-design	**22** ecd
… …	
Thread support	**87** ecd
total	***317 ecd***

Status: *Capacity:* **317 effective coder-days**
Requirement: **317 effective coder-days**
Delta: **0 effective coder days**

simple SaaS horizon plan

Horizon:	Dates:	Jul.1—Dec.1
	Workdays:	104
	Coding Factor:	0.75
	Coding Days:	77
	Sprints:	5
Capacity:	*days available*	
	Fred	**31** ecd
	Lorna	**58** ecd
	…	…
	Bill	**47** ecd
	total	***317 ecd***
Requirement:	*days required*	
	AR report	**14** ecd
	Dialog re-design	**22** ecd
	… …	
	Thread support	**87** ecd
	total	***317 ecd***
Status:	*Capacity:*	**317 effective coder-days**
	Requirement:	**317 effective coder-days**
	Delta:	**0 effective coder days**

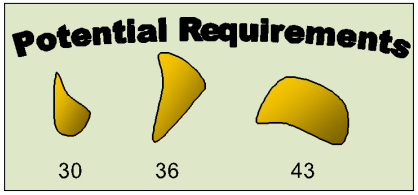
sizing available resources

- who can work on the release?
  - skills & familiarity required
- for how long?
  - count of workdays in development phase (horizon)
  - is each resource (developer) available for the entire development phase?
  - are they available 100% or are working on other projects too?
  - subtract (estimated, where necessary) vacation

sizing available resources (2)

- how much time can the developers spend actually writing code?
  - work factor = *w*
  - converts 8-hour (nominal, arbitrary) days to time available to write code and unit tests for the next release (or horizon)
  - ex. $w = 0.6 \Rightarrow 0.6 \times 8$ h/d = 4.8 h/d
  - first estimated, then measured quantity
  - accounts for things like:
    - sick days, other tasks, meetings, etc.
  - for a "normal" developer is usually around 0.6

sizing potential requirements



- cost / benefit analysis
  - cost: financial + opportunity
- sizing in ECDs
  - planning poker: Inherent size of the work item
  - who will work on it? resize
  - productivity of that person (*w*)
- ensure that units are well understood

the capacity constraint

- after all is done in a release (horizon)…

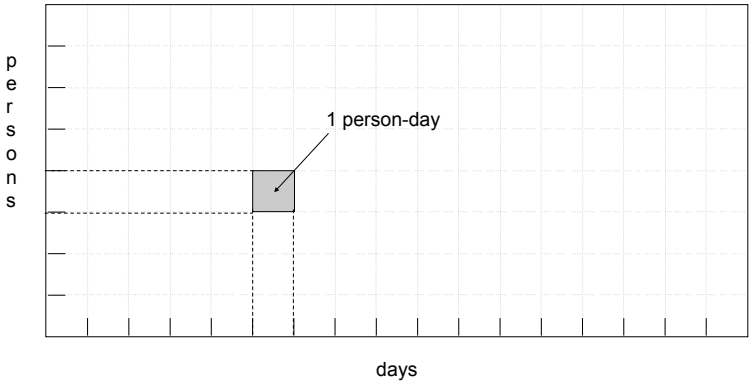
  actual resources used == sum of actual feature time

- this is always true no matter what, so it really is a *constraint*

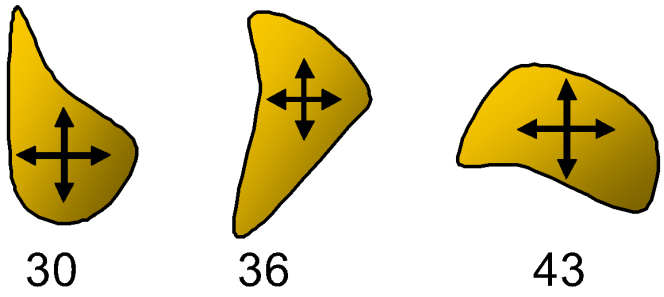
- so, given that we know this must work out for each planning cycle, we estimate both sides and force them to be equal

  resource estimate == sum of feature estimates

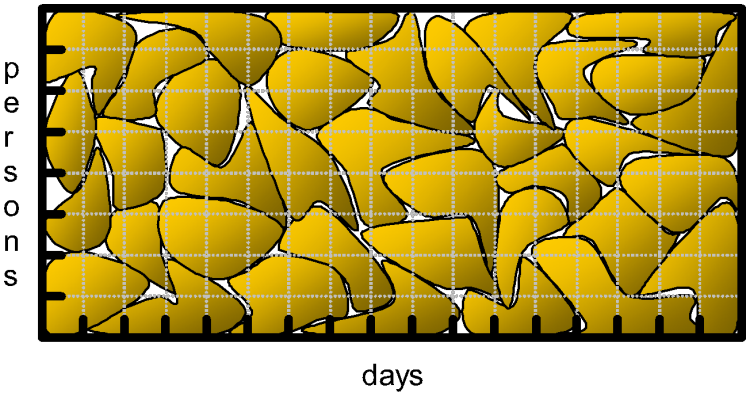
geometric analogy – capacity



geometric analogy – requirement



geometric analogy – capacity constraint



everything must fit!

planning

- what are we building? **F**
- when will it be ready? **T**
- how many developers? **N**

$$F \leq N \times T$$

- plan must respect the capacity constraint
- must continuously update the plan to maintain this property

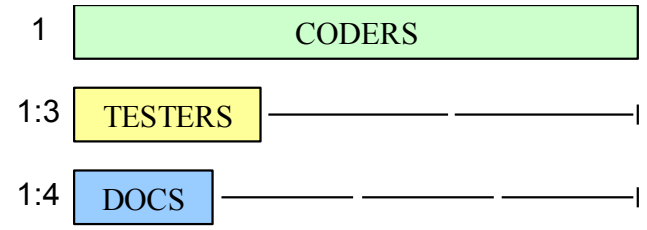
planning non-coding activities

- in horizon planning we explicitly plan coding activities only
  - other resources: testers, docs, managers
  - other phases: spec., test, etc. (non-coding)
  - above sized relative to coding phase/resource
- why?
  - debugged code is ultimate target – can't ship feature set if it's only 90% done for example
  - how much time to devote to docs, testing, spec?
  - when is enough, enough?

planning non-coding activities (2)

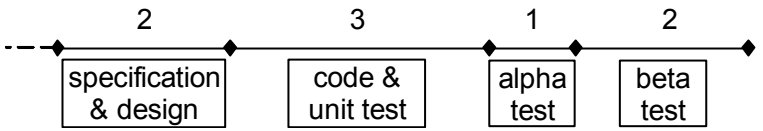
- how?
  - establish ratios
  - measure what works for ratios for a given product
  - adjust next time around
  - converges rapidly
  - initial guess is usually pretty good

resource ratios



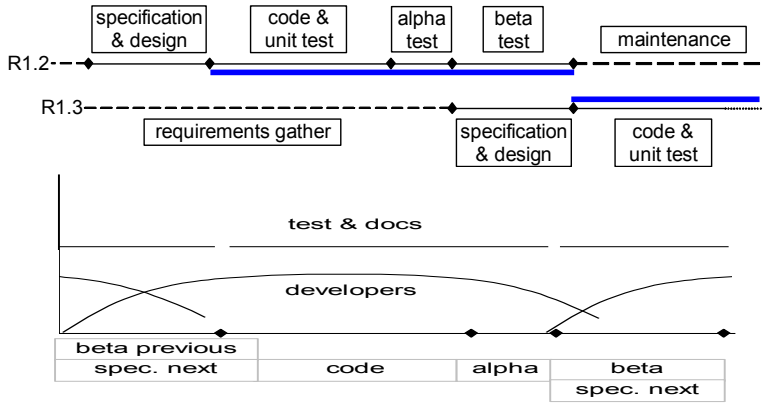
- typical ratios used in horizon planning
- adjust as necessary
- assumes availability throughout the (overlapping) release cycle.

traditional phase ratios



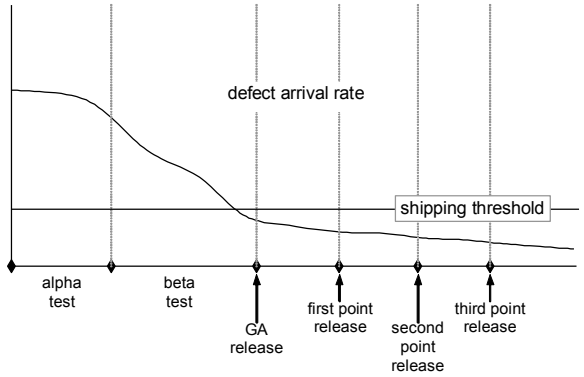
- typical ratios used for shipping software using traditional practices
- adjust as necessary
- if performing extensive automated unit testing during coding phase (possibly utilizing TDD) , test phases can be considerably reduced (5:1)

traditional release overlap



- overlapping release cycles smoothes resource utilization

shipping the traditional release



- after dcut, proactive management is gone
- can only watch defect arrivals and hope for the best.
  - if your ratios are way off you could be in trouble and not know until it's in the field
  - react by adjusting them for next time (hope there is a next time!)

SaaS coding ratio

- use a ratio of:

predominantly coding days (PCDs)
to
workdays in the planning horizon

- one definition of a PCD may be any day where a coder spends > 1 hour coding features in the next release
- defects should be managed at every sprint, and a stabilization sprint inserted when the levels are too high.