Computer Science
UNIVERSITY OF TORONTO

lecture 19
software development planning continued…

csc302h

winter 2014

Computer Science
UNIVERSITY OF TORONTO

recap from last time

- CMM levels & tasks:
  1) initial
     - totally ad-hoc, no process
  2) repeatable
     - planning & tracking
     - project management
  3) defined
     - process definition, reviews, etc.
     - engineering management
  4) managed
     - quality of process & feedback for improvement
     - quantitative management
  5) optimizing
     - continuous improvement & change management

Computer Science
UNIVERSITY OF TORONTO

recap from last time (2)

- how the top-10 relates to ISO 9000 and to the CMM levels
- started talking about planning
  - what goes wrong if you don't plan
    - crossing the chasm
  - why plan? – external pressures
- planning essentials
  - what are we building?
  - by when will it be ready?
  - how many people do we have?

Computer Science
UNIVERSITY OF TORONTO

recap from last time (3)

- the essence of planning is uncertainty
  - react to changes; both internal & external
- the difficult question is:

  what are we building?

  by when will it be ready?

  how many people do we have?
- can we do all three at once?
- a common problem is to answer the three questions, but not the difficult one
  - good planning to avoid the death march

Computer Science
UNIVERSITY OF TORONTO

Software Development Planning II

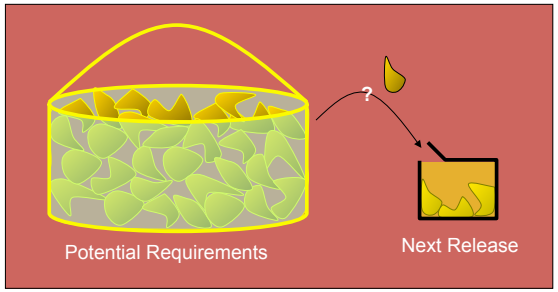
Computer Science
UNIVERSITY OF TORONTO

today

software development planning continued…

Computer Science
UNIVERSITY OF TORONTO

eliciting potential requirements



- starts with a wish-list
- stated as business requirements
  - features and/or architectural enhancements

Computer Science
UNIVERSITY OF TORONTO

A 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**

Computer Science
UNIVERSITY OF TORONTO

sizing available resources

- who can work on the release?
  - skills & familiarity required
- for how long?
  - count of workdays in development phase
  - 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

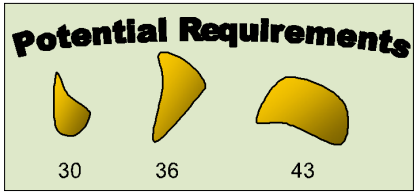
Computer Science
UNIVERSITY OF TORONTO

sizing available resources (2)

- how much time can the developers spend actually writing software?
  - 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

Computer Science
UNIVERSITY OF TORONTO

sizing potential requirements



- cost / benefit analysis
  - cost: financial + opportunity = person days
- 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

Computer Science
UNIVERSITY OF TORONTO

the capacity constraint

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

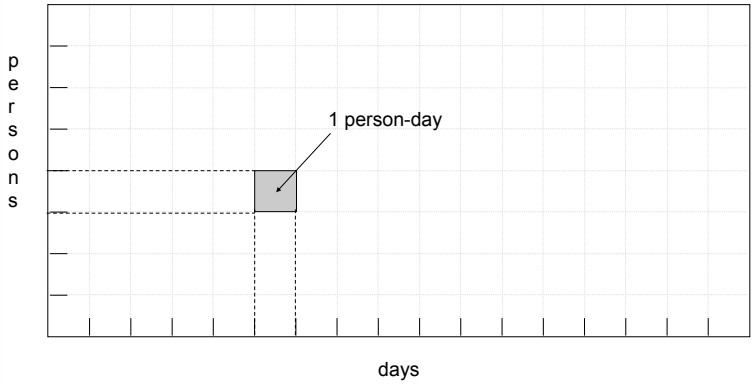
  actual resources used == sum of actual feature time

- 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

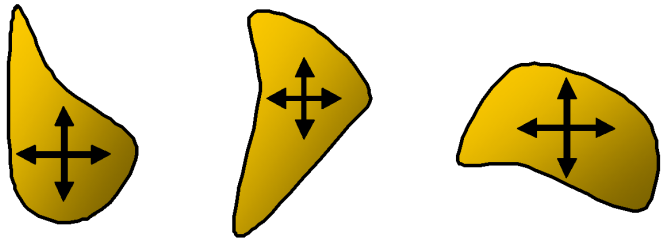
Computer Science
UNIVERSITY OF TORONTO

geometric analogy – capacity



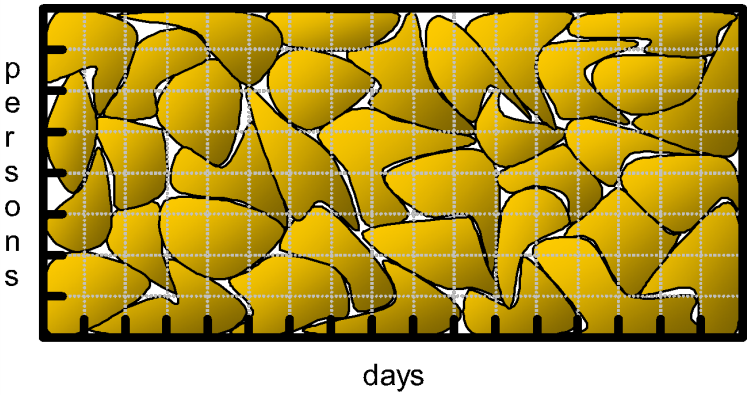
Computer Science
UNIVERSITY OF TORONTO

geometric analogy – requirement



Computer Science
UNIVERSITY OF TORONTO

geometric analogy – capacity constraint



everything must fit!

Computer Science
UNIVERSITY OF TORONTO

release planning

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

$$F \leq N \times T$$

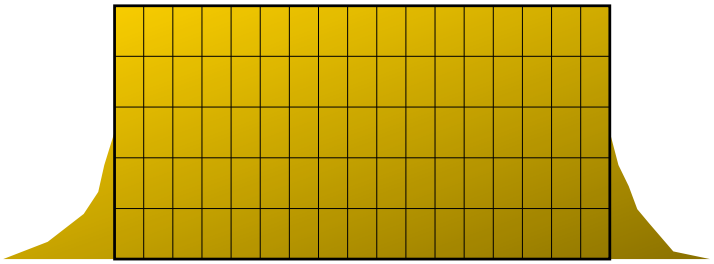
- plan must respect the capacity constraint
- must continuously update the plan to maintain this property

Computer Science
UNIVERSITY OF TORONTO

most common problem

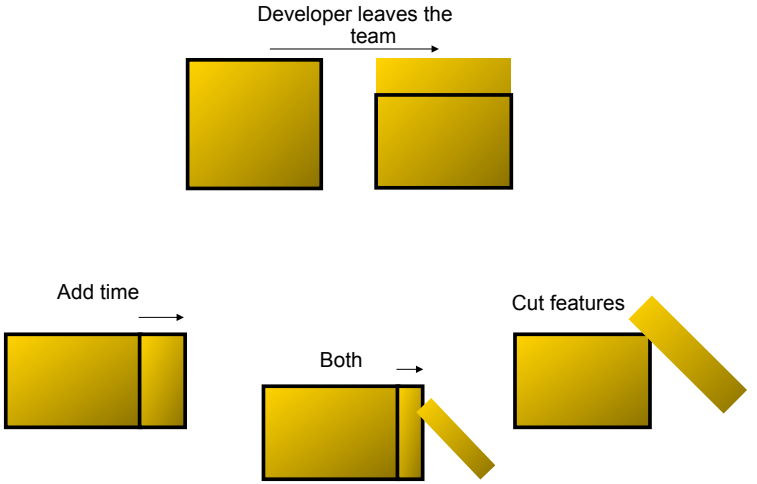
- comes from either:
  - not knowing, or
  - knowing but hoping for the best (death march)

  (can happen at the start, or later in the plan)



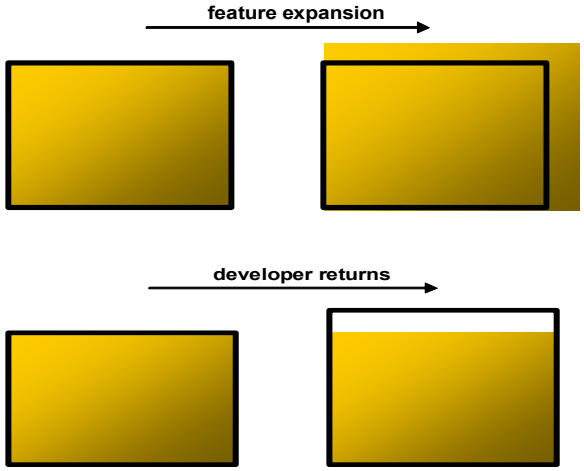
Computer Science
UNIVERSITY OF TORONTO

dealing with overflow



Computer Science
UNIVERSITY OF TORONTO

dealing with overflow (2)



Computer Science
UNIVERSITY OF TORONTO

organizational issues

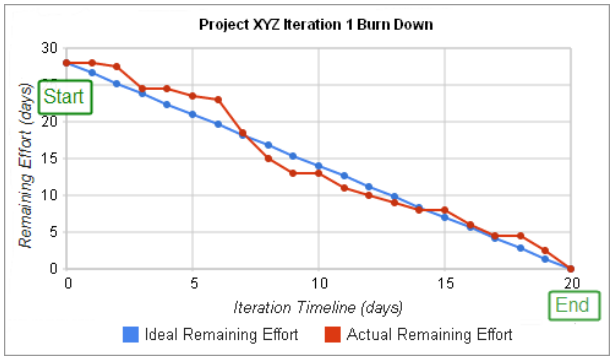
- management must appreciate that software development carries with it certain inherent risks.
- the business of a software organization is to manage and adapt as possibilities continuously become reality.
- ranting and raving is unproductive.
- with good data, good managers can make good decisions.

Computer Science
UNIVERSITY OF TORONTO

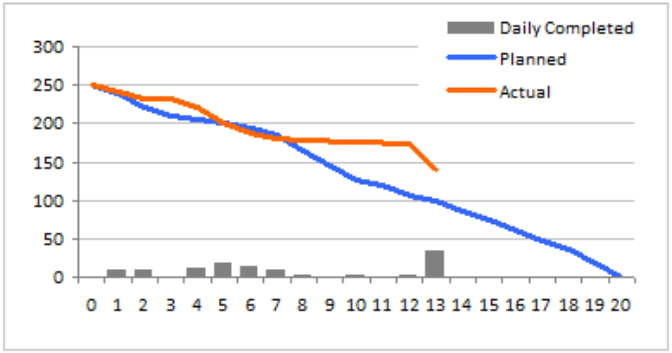
bonus material…

burndown charts

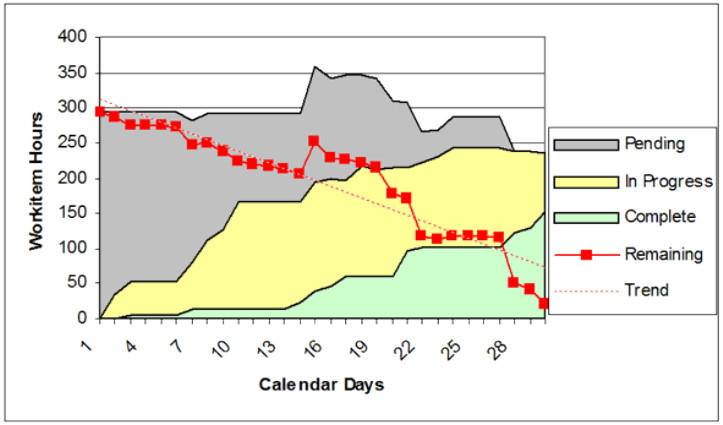
burndown charts



burndown charts (2)



burndown charts (3)



- wtf is that!?!?
  - Trust me, don't show this one to your CEO!