Lecture 10: Tracking

CS190: Software System Design

February 25, 2002
Steven P. Reiss

I. Today's Class

A. Why do tracking and planning

B. What to track

C. How to plan

D. Techniques for quality control

II. Planning

A. Classical project planning

1. **Assume you have the whole design**
2. **Determine how long each part will take**
3. **Determine the number of personnel**
4. **Construct a pert chart and find the critical paths**
5. **Construct a time chart indicating who does what when**
6. **Does this work -- why or why not**

B. Planning in an XP project

1. **You can't do this in XP -- no design in advance, specifications are changing, ...**
2. **Instead you want to plan the sequence of releases**
   a) A release incorporates a set of features represented by stories
   b) You want to determine which stories are in which release
3. **This is typically done on an incremental basis**
   a) This allows new stories to be added dynamically
   b) It allows taking into account changing priorities
   c) It allows taking previous experiences into account
   d) Objective is to plan the next release and to modify the plan on the current release dynamically

4. Determining the next release

- a) Choose a set of features to be included
  - (1) These can either be stories
  - (2) Or they can be significant code changes (refactorings) determine as necessary but not yet done
  - (3) Or they could be test cases that fail.
- b) You have to priorities these
  - (1) Better to get 7 of 10 done than to get 70% of 10 done.
  - (2) You want to avoid conflicts among developers if possible
  - (3) Might have to break features into smaller pieces and get the pieces up first
  - (4) There might be dependencies among features
- c) Assign pairs to work on one or more features
  - (1) Based on estimated cost of feature
  - (2) Based on quality & abilities of the pair
  - (3) Based on who has the most time each week
  - (4) Team should complete one feature before moving to next

5. Steering the current release

- a) Why change plans
  - (1) Estimates are often wrong
  - (2) Some things will require more time, cause more problems, etc. than expected
  - (3) You have to adapt to this
  - (4) People have problems too
- b) You want to have a release on the release date, not a non-functional system
  - (1) Might need to reassign people, change priorities
  - (2) Might need to drop features until later release

C. How to do the planning

1. One key is having the necessary information

a) This is where tracking comes in

2. Another is to get good at cost estimation

a) This will come with experience, both in general and with your particular system

III. Tracking

A. Tracking the Process

1. You need to know where you stand wrt project, release, etc.

2. You want to track resources (people, ...)

3. You want to track something tangible

a) Keep track of the number of stories

(1) Total, done, yet-to-do

(2) At various different priority levels

(3) Keep graphs/tables/spreadsheets of this information over time with a history

b) Keep track of the software

(1) Number of classes, methods, LOC

B. Tracking Quality

1. You want to know how close to working your system is

2. You want to know what parts of the system are having problems

3. Again you want to track something tangible

a) Keep track of your test cases

(1) Number of tests, number passed, number failed

(2) Do this on a per-package basis (or per-class)

(3) Use to identify weak points in the system

(4) Again maintain this over time and keep a history

b) Keep track of bugs

C. Tracking Bugs

1. As you work and test the system you will discover that things don’t work perfectly

a) This should come from test cases

b) But might also show up as new stories

c) Or, if simple, as a note to fix or change something

2. Code defensively to help identify problems early on

a) Make all assumptions about arguments, etc. explicit

b) Make heavy use of strong typing throughout

(1) Enumerations in Java

(2) Templates in C++

c) Check return values for validity

d) Use exceptions for errors where possible

3. Its important to have a simple, easy to use, mechanism for recording and tracking these problems

a) You want to encourage people to note the problems

b) You want to make it easy to discover what been noted and what hasn't

c) You want to make it easy to get a handle on what needs to be done

4. Techniques

a) Simple techniques -- TODO lists

(1) Can be a global todo list or each individual maintains his/her own

(2) These need to be merged and then sorted

(3) Sort by package, type, etc -- depending on how you want to fix them

b) More complex techniques

(1) Bug databases -- keep a database of open issues

(2) Let programmer query that database by various properties

(3) Let users or programmers enter new elements

(4) Several commercial systems

(5) Also freeware systems -- gnats for example