csc444h: software engineering I

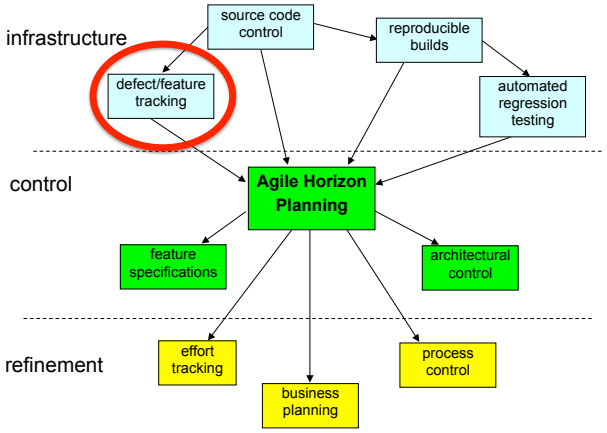
matt medland
matt@cs.utoronto.ca
http://www.cs.utoronto.ca/~matt/csc444

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

defect tracking

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

top-10



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

defect tracking

- keeping track of all the defects that have been discovered
- keeping track of all the steps required to validate, correct, and take preventative action for a defect
- necessary:
  - to not lose any reported defects
  - to co-ordinate defect resolution
  - to ensure coders don't work on non-defects
    - features masquerading as defects
    - wasting time fixing something that isn't broken
    - wasting time chasing down a badly reported defect
  - to control defect correction activity
    - ensure the right defects are being worked on
- in practice:
  - A database of defect records
  - A workflow driven by the **state** and **owner** fields.

defect information

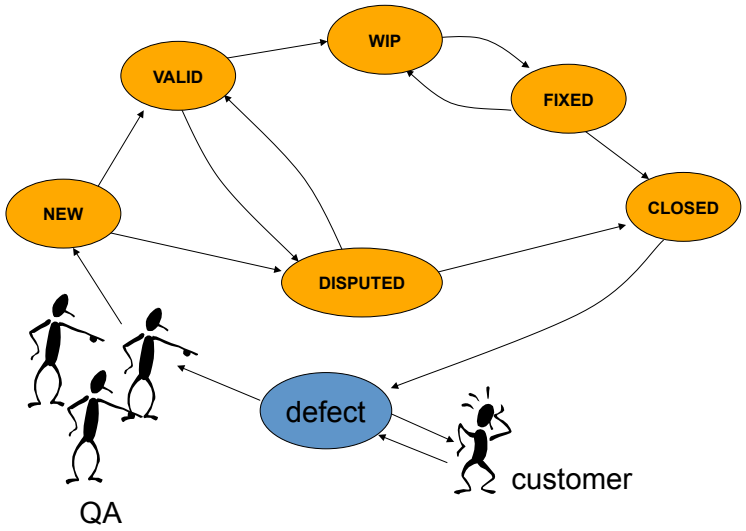
- where it was found
  - product, release, version, hardware, os, drivers, general area
- who found it
  - customer, internal, when
- description of the defect
  - summary, description, how to reproduce, associated data
  - links to related defects or features
- triage
  - severity, likelihood → priority
- audit trail
  - all changes to the defect data, by whom, when
- state
  - state, owner

priority matrix

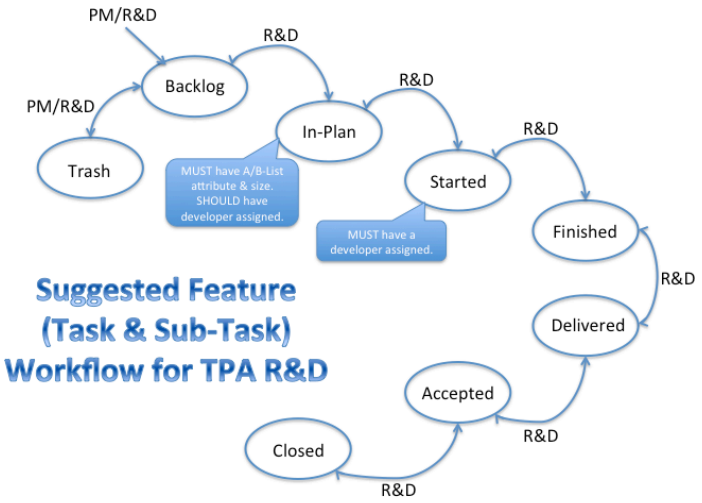
severity \ likelihood	low	medium	high
crash, bad data	2	1	1
work around	5	3	2
cosmetic	5	4	3

- submitter of defect chooses severity and likelihood
  - may later correct if determined to be an exaggeration or in error
- priority assigned according to the priority matrix
- humans may change the priority using their judgment
  - no need to stick to "the matrix", which is after all too simple to account for every contingency

defect workflow



issue workflow (also used for bugs)



developer assignment

- bug is auto-assigned to a developer based upon
  - "auto" may mean assigned by a person
  - the product in which it was found
  - the functional area of the defect
- catch-all category (misc.) goes to team-lead for defect assignment and overview for assignment elsewhere.
  - keeps track of the defect load by priority on all coders
  - balanced the load
  - chips in where needed
- developers may move the defect to the appropriate coder without management permission.
  - may also move to team lead for re-assignment
  - natural corollary to auto-assignment.

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

management controls

- provide defect visibility to enable management to ensure defects are appropriately prioritized
- management must:
  - review all active defect records
  - ensure priorities are appropriate
  - if languishing too long in a given state, act
  - ensure coders are working on defects of appropriate priority at any given time
- system support
  - most systems can be configured to
    - send e-mail and/or re-assign to manager when certain conditional action thresholds are reached
      - ex. priority 1 defect with state unchanged for 24 hrs.
    - post daily reports of overdue defects

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

controls on the system

- most defect tracking systems allow permissions
- each user is given various group memberships:
  - developers, testers, managers, builders, …
- permissions can then be set up by
  - group, state, field
- don't do it!

  Q: what are you trying to control?

  A: source code

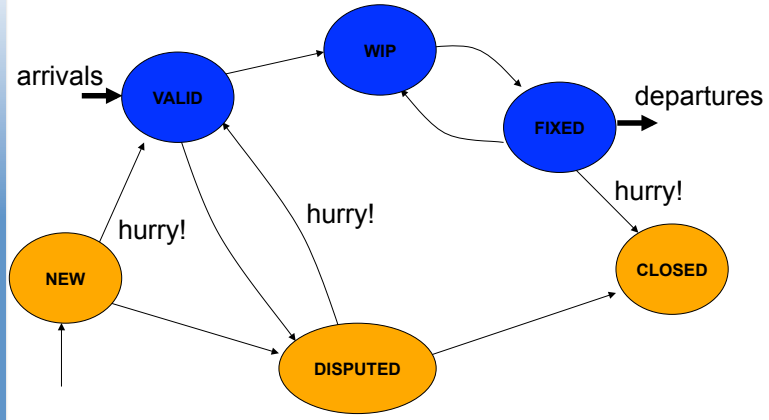
- putting restrictions on defect control system will not help you to gain control of the source
  - it will hurt
  - developers will work around silly security restrictions
  - defect system will not accurately reflect what is being worked on
- dirty data will go uncorrected

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

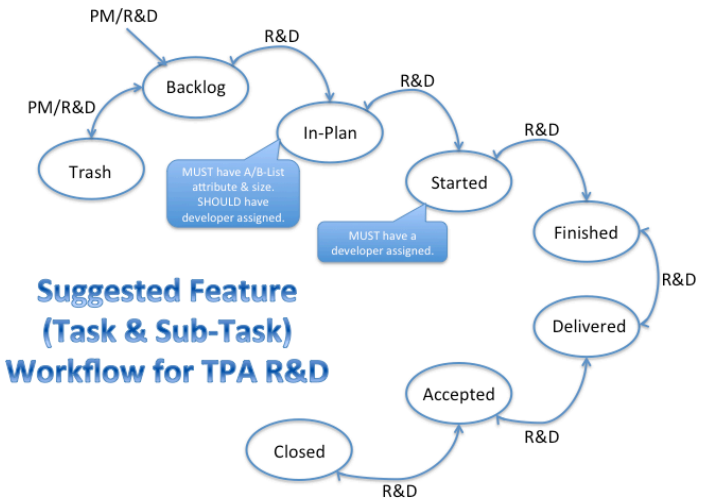
metrics

- proper defect tracking enables the gathering of good, clean defect arrival/departure data.
- gives insight into productivity of
  - developers fixing defects
  - testers finding defects
- clean data is essential
  - ex. if no way to validate defects
    - lots of arrivals may be due to bad code or to bad defect triage
    - may expend a lot of effort on coding initiatives and numbers will go the wrong way!
  - must quickly get defects out of **NEW** and **FIXED**
- arrivals:
  - defects per day entering into **VALID**
- departures:
  - defects per day going from **FIXED** to **CLOSED**
- total:
  - sum of defects in states **VALID**, **WIP**, and **FIXED**.

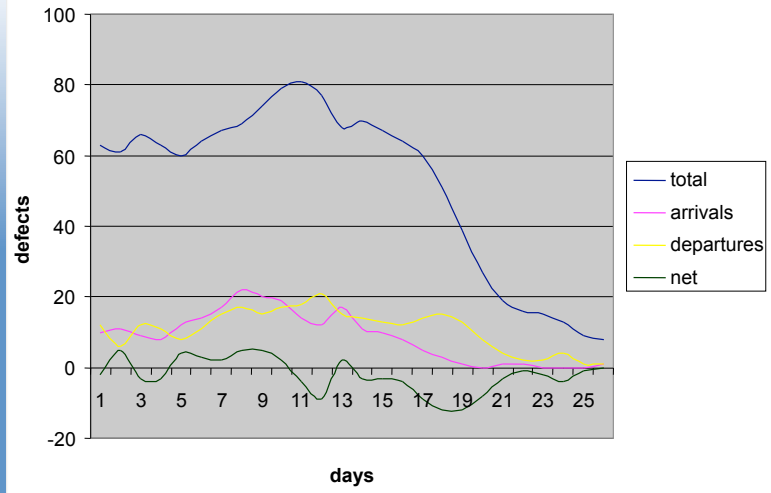
metrics (2)



issue workflow (also used for bugs)



metrics (3)



towards release

- these metrics should be tracked:
  - by product
  - by priority
- company should establish shipping thresholds
  - ex. no known priority 1 or 2 defects
  - ex. arrival rate for priority 1-3 < 1 defect per day
- watch trends, compare to last release & if not good:
  - try the “bug olympics”
  - “bug blitz weekends” and/or stabilization sprint
  - slip the release date
  - clean up the architecture

relationship to source control

- two reasons for changes to source:
  - fix a defect
  - add a feature
- link source control and defect/feature tracking
- whenever a coder checks in a change
  - prompted for: defect or feature ID
  - check to ensure assigned to them
  - persistently stored
- this allows management to see
  - what was changed (see diff report)
  - why it was changed (look up feature/defect description)
  - by whom
- is this really control?
  - yes: audit trail

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

source control report

Last 24 hours

	David	Kathleen	Douglas	Brian
D100203	23			
F100350		108	34	
D155401			56	
D100343				10
D100453	1			
F100782			508	
Totals:	**24**	**108**	**598**	**10**

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

defect attribution

- beginning to understand what are the systemic root causes of defects.
- include as data in the defect tracking system that must be there before defect is closed
- should record time taken to deal with it, or at least a “difficulty” field (high, medium, low)
- attribute to:
  - where in the source code
    - can identify modules whose re-design will add most bang-for-the-buck
  - which developer introduced it
    - organizationally tricky but very useful
  - during what phase
    - spec, design, code

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

customer issue tracking

- distinct from defect tracking
- customers have many issues:
  - how to use software
  - installation issues
  - perceived problems
  - problems that have already been resolved in a previous patch
  - known issues
  - ship me a manual, please
  - …
- some of these issues will result in new defects
- requirements of issue tracking systems will include:
  - customer relationship management tie-in
  - searchable knowledge bases
  - customer tracking of issue progress
  - …

shipping with known defects

- 0-defects is not practical or sustainable for most businesses
  - how many defects are acceptable?
  - how many are you shipping?
    - defect seeding
      - inject defects, see how many are found, use the ratio
      - hard to work this in practice
- must measure customer satisfaction with perceived level of defects and correlate to known defects at ship. ex.
  - if we ship with 350 known defects and customers are down on the release then it's too high
  - if we ship with 50 and customers say "best release ever" super stable, then it's good.
    - might want to use 50 as the shipping threshold, and then gradually lower that over time

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

adjusting development/test ratio

- can only compare across releases if have a consistent testing effort
  - same number of testers, same productivity, same time, same general size of the release
- if increase size of testing team relative to coding team,
  - ratio of known to unknown defects decreases
- assume ratio is 50%
  - ship with 50 known, actually shipping 100 defects
- add testers, raising ratio to 75%
  - ship with 75 known, actually shipping 100 defects
    - good to know. if increasing testing effort without increasing coding efforts, will be hard-pressed to meet the old thresholds
- add developers, lowering ratio to 25%
  - ship with 25 known, actually shipping 100 defects
- add developers and testers
  - ratios stay the same
  - but will reach the thresholds faster for the same sized effort

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

release notes

- when shipping point releases, good to say which defects are fixed
  - hard to get this info!
- start with source control and defect tracking to see which defect corrections have been checked in since the last point release
- must describe the defect in terms the users will understand
  - ex. load this data file it crashes
    - good enough to find and fix the defect
    - not good enough for release notes
      - must track down the root cause, and extrapolate into what kind of situations will trigger the defect.
      - If doing this, must make it a part of the defect correction process

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

automated patching

- ability for the software to query a server to see if it is up-to-date
  - if not, then download an appropriate, ideally small, patch and apply it
- distinguish "critical" from "optional"
- run immediately after install
- facility must be able to chain patches
- determine smallest download combo to get you from where you are to current version
- need excellent build/release disciplines to ensure release numbers completely identify the file set
  - will want to provide binary diff files as patches – need to be sure
    - will double-check a checksum on all files before applying anything!

automated patching (2)

- a patch always starts with a complete image of the software installed into the file system.
  - a normal, regular release
  - test it as such
- use a patching utility to generate a binary diff patch
  - point at release A and release Z
    - will generate a small patch self-installer that moves you from A to Z
  - point at releases A, B, C and Z
    - will generate a somewhat larger patch self-installer that is capable of moving the software form any of the releases A, B, or C to Z
      - larger, but saves due to common files between A, B, C
      - if no common files, is a waste
  - end user may have to download:
    - patch 1: from A to W
    - patch 2: from W to Z

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

feature tracking

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

feature tracking

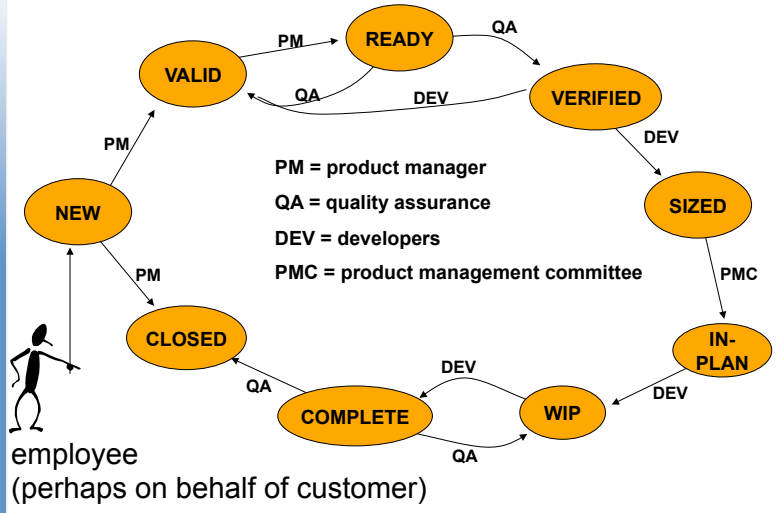
- keeping track of all the features that have been requested
- keeping track of all the steps required to validate, specify, design, code, and test each feature
- necessary:
  - to not lose any requested features
  - to co-ordinate feature addition
  - to make it clear which features are in and which are out
  - to ensure only approved features get worked on
- in practice:
  - a database of feature records
  - a workflow driven by the **state** and **owner** fields

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

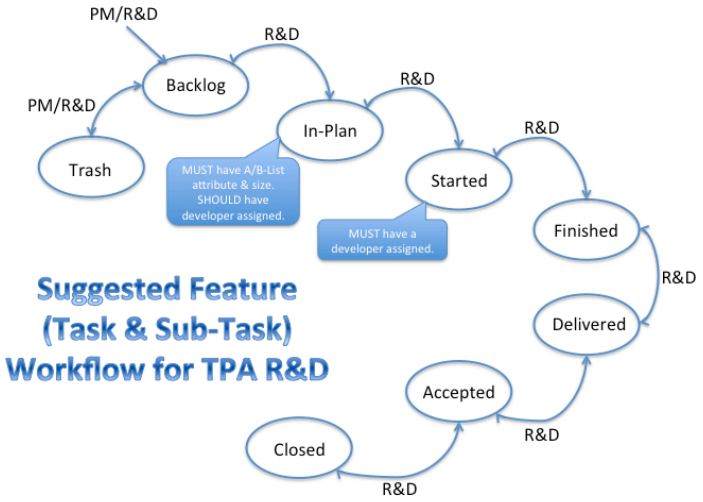
feature information

- description
  - one phrase summary, one-paragraph description
  - which product, which area of the product, targeted at which segment?
- who requested it
  - customer, internal, when
  - internal champion
- priority
  - customer desired priority
  - company assigned priority
- target release
  - set once in a release plan
  - set if decided definitely not in the next release
- effort
  - # of ECDs required to implement the feature
- attached documents
  - specification, design, review results, ...
- working notes
  - time stamped notes forming a discussion thread
- process tracking
  - spec required? spec done? spec reviewed? ...

feature workflow



feature workflow (2)



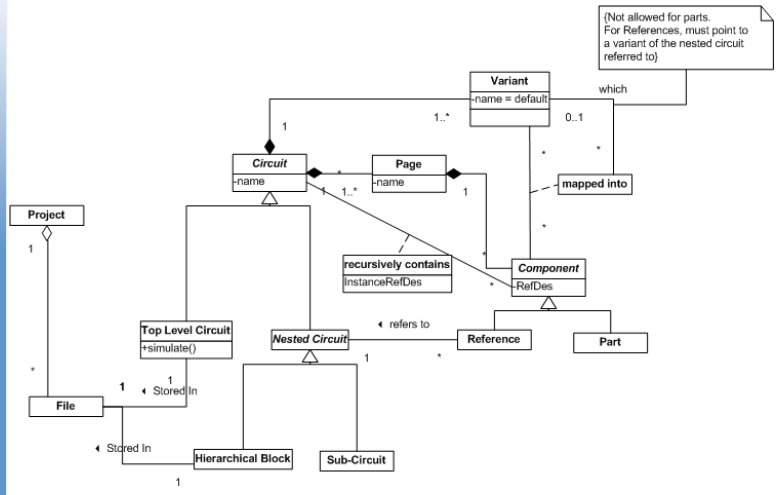
specifications

- after features are IN-PLAN
  - dev/docs/QA/PM meet to discuss each in-plan feature
    - will likely bundle a bunch of lower-level features together
  - will discuss and scope out feature
    - notes attached to feature record
  - will decide if a detailed written spec is required
    - boolean field set accordingly
- specification documentation
  - describes all externally visible behavior of the feature
    - does not discuss internal design considerations
    - rough (conceptual) design for UI (ex. mockups in balsamiq)
    - menu items, options, what they do
    - algorithms
    - impacts to other areas
      - extend reports? files? databases?
    - compatibility concerns

UML for analysis

- specifications will often use UML to clarify the concepts that are being discussed
  - name concepts unambiguously
  - show how they are related
  - get everybody to a common understanding
- UML diagram
- explained with written text
- then go on and describe the feature

example UML



specification review

- once a specification is written, should review it
- get a group together
  - mainly developers
- have them read the spec
- ask them to come to the meeting prepared
- does not review what is the feature or how the feature is exposed in the software
  - too late for that – should have been discussed in requirements validation and spec meetings already
- identify defects in the spec:
  - incompleteness
  - inconsistency

example spec review results

variant spec review meeting – Multisim V8
chair: Braulio
scribe: Dave
reviewers: John, Maks, Rodney, Anita, Shauna
feature suggestions:
change name of all recursively mapped-to variants ***
allow export to UB of selection of variants ***
sight-click menu to change active variant? ***
don't silently automap? ***
when hierarchy viewer gone – must UI indicate active variant? ***
minor issues:
delete all variants?
across circuits not specified?
preferences circuit tab should also have show variant status attribute
defects:
no mention of multi-section components?
missing detail: if printing will it print dimmed as you see it on the screen?
netlist report format changes?
Ultiboard V7 compatibility issues not addressed
refdes mapping when using instance refdes and variants?
[see Dave's spec and Anita for how instance refdes will work in V8]

other reviews

- feature review
  - pre-spec: is used to ensure feature is well-formed
- design review
  - at least by chief architect
  - similar to a spec review
- code review
  - informal: another developer looks over the code, or
  - formal: meeting
- feature demo
  - make a point of demoing every feature as soon as it can be
  - scribe should record actions (scribe dictates pace)
  - good early milestone

effort tracking

- track time:
  - dedicated hours spent on each feature
  - dedicated hours spent fixing defects
  - vacations taken
- need a system:
  - fine-grained time-tracking system
  - will prompt you with features you are working on (in WIP state)
- no need to track all time
  - may be counter-productive
- combine with a prompt for a re-estimate each time time is logged against a feature
  - prompts for reason if slips

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

management control

- developer work factors and vacation estimates
  - managing them
- actual versus estimated feature time
  - managing them
- progress to process