Week 9

COMP 370/470: Software Quality and Testing

Plan du jour

- Test 1 discussion
- Quality management models
- In-process metrics for software testing
- Group activity

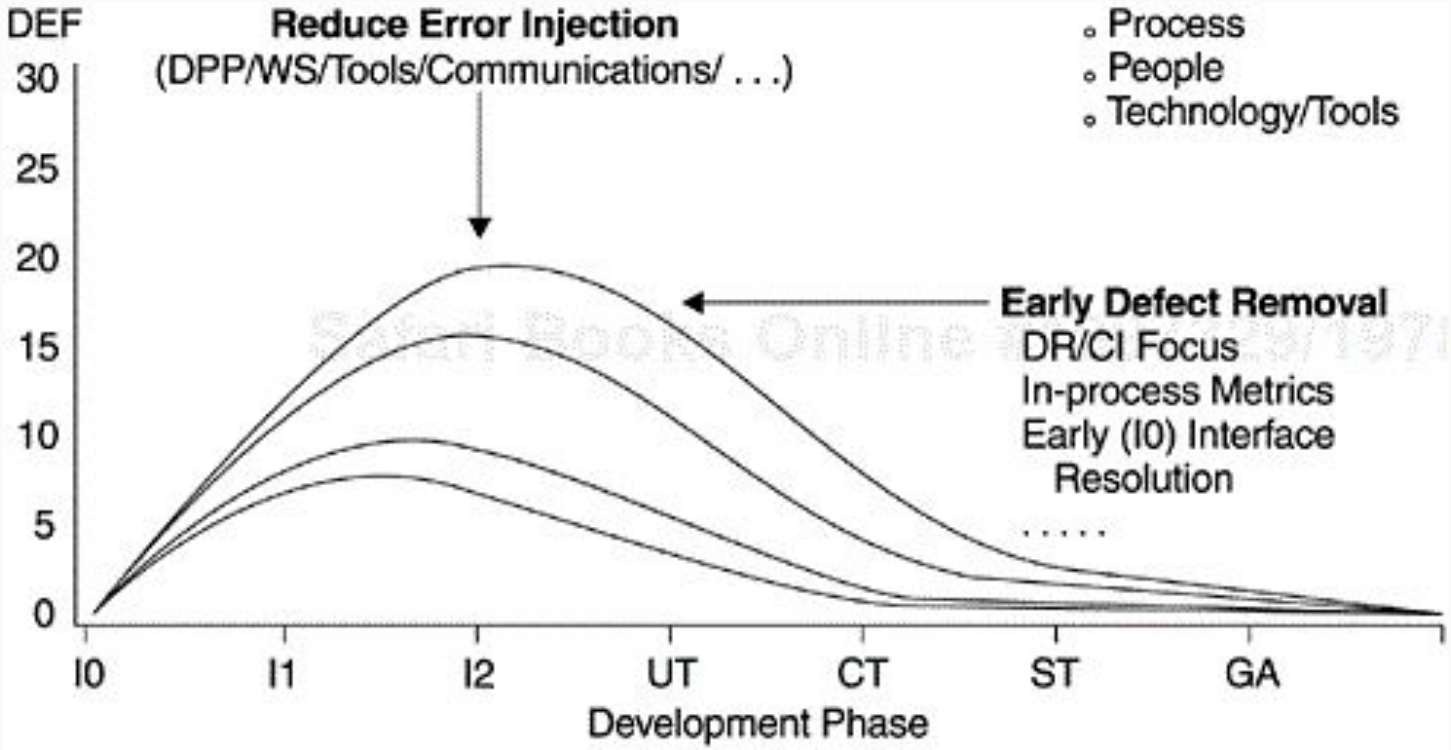
Reliability estimation

- Assess product quality
- Project number of defects
- Estimate mean time to next failure (post-release)
- Discussed in week 7 (chapters 7 and 8)
- Focus on “small q”

Quality management

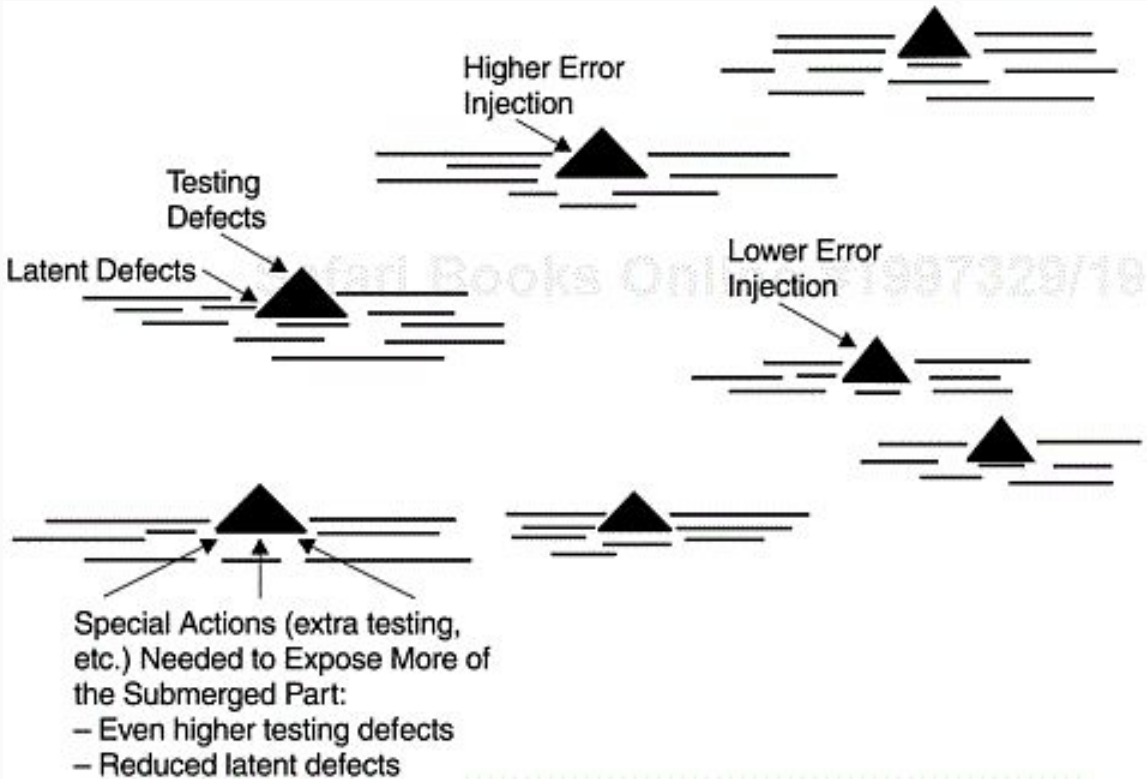
- Recall progression: QC, QA, QE/QM
- Monitor and manage software quality during development
- Focuses on “Big Q” especially process quality
- Overarching principle: *Do it right the first time* (pre-release)
  - Error prevention
  - Early error detection through design reviews and code inspections
  - Later detection through thorough testing
- Can still use Rayleigh model

Rayleigh model applied to development quality improvement

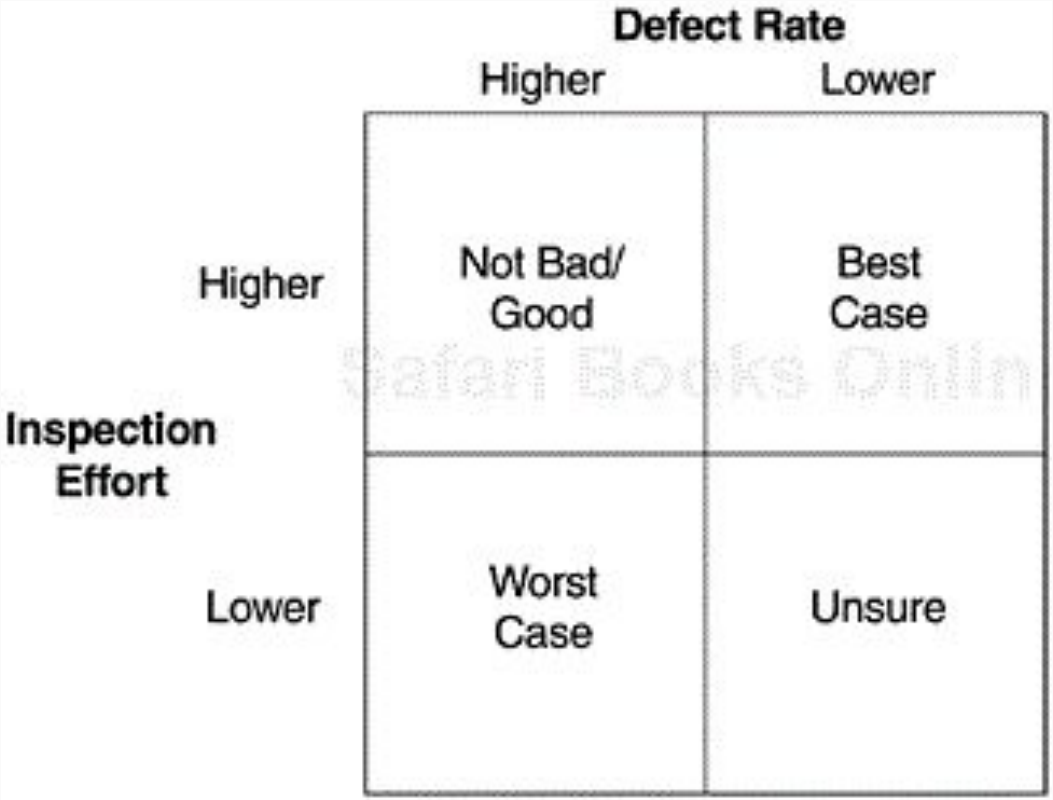


DPP: defect prevention process
WS: working sessions?
DR: defect removal
CI: code inspection

Iceberg analogy for pre-release vs. post-release defects

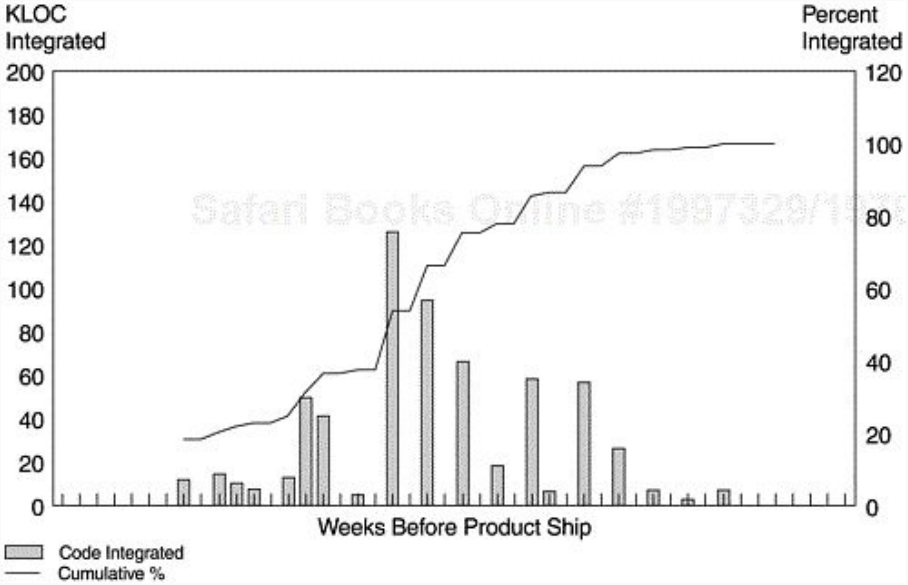


Simplified inspection effort vs. defect rate metric



Software code integration pattern

- Can use to track code completion and integration progress
- Can distinguish healthy and unhealthy (back-end loaded) patterns
- Can compare among projects and releases



Other useful models for in-process quality management

- PTR (problem tracking report) submodel - for tracking
- PTR arrival and backlog projection - will it decrease toward release date?
- Reliability growth models (discussed previously)

Sample inspection scoring checklist (using 10-point Likert scale)

- Design
- Work meets requirements
- Understandability of design
- Extensibility of design
- Documentation of design
- Effectiveness of this inspection
- Does another inspection need to be held?

- Code Implementation
- Work meets design
- Performance considerations
- Understandability of implementation
- Maintainability of implementation
- Documentation
- Effectiveness of this inspection
- Does another inspection need to be held?

Orthogonal defect classification

- Function
- Interface
- Checking
- Assignment
- Timing/serialization
- Build/package/merge
- Documentation
- Algorithm

Addl. attributes when defect is opened

- Activity
- Trigger
- Impact

Addl. attributes when fix becomes known

- Target
- Defect type
- Defect type qualifier
- Source
- Age

In-process metrics for software testing

- Test progress S-curve: planned, attempted, actual
  - linkedin.com/pulse/s-curve-track-project-progress-h-alan-karatas
- Testing defect (PTR) arrivals over time
- Testing defect (PTR) backlog over time
- Product size over time
- CPU utilization during tests
- Unplanned “initial program loads” (reboots/restarts) after crash or hang
  - MTTF/MTBF
- 2x2 effort/outcome matrix (as before)

When is the product good enough to ship?

- Stability, reliability, availability
- Defect volume/density
- Outstanding critical problems (“showstoppers”)
- Feedback from early customer access programs
- Other relevant quality attributes

May be able to perform assessment of these indicators compared to prior releases using traffic light (green-amber-red)

Group activity:
Mining software repositories (MSR)

- github.com/klaeufer/ghedutils small example of GH REST API calls and drilling into resulting JSON data using jq
  - *Make sure your API requests are authenticated to avoid tight rate limits!*
- Better to use a proper client binding for the API
  - github.com/PyGithub/PyGithub more conventional
  - ghapi.fast.ai looks very good - see also github.blog/2020-12-18-learn-about-ghapi-a-new-third-party-python-client-for-the-github-api
  - Alternative: Google AppScript www.benlcollins.com/apps-script/oauth-github
  - Alternative: Python www.analyticsvidhya.com/blog/2020/07/read-and-update-google-spreadsheets-with-python
  - Alternatives: JavaScript github.com/octokit/rest.js, Java, Scala, etc.
- Idea: build an automated data pipeline
  - Pull data from the API - *keep pagination in mind*
  - Import into Google sheets or Python notebook
  - Mine, chart, etc.

Group activity: Stage 1: issues

- Pull ***all*** issues (open and closed) for the given owner (user or org) and repo:
  - docs.github.com/en/rest/reference/issues#list-repository-issues
- Request parameters
  - page, per_page - see docs.github.com/en/rest/overview/resources-in-the-rest-api#pagination
  - state=all
- Identify attributes of interest
  - state
  - labels - array, drill into string values
  - assignees - array, take size
  - comments - array, take size
  - closed_at
  - created_at
  - locked
- Visualize and analyze in various ways
  - Number of open and closed issues by any time unit (day, week, month, year) over any period (YTD, year, lifetime of project)
  - Do issues with more assignees stay open longer?
  - Do issues with more comments stay open longer?
  - ...

Group activity: Stage 2: commits

- Pull ***all*** commits for the given owner and repo:
  - docs.github.com/en/rest/reference/repos#list-commits
- Request parameters
  - page, per_page - see docs.github.com/en/rest/overview/resources-in-the-rest-api#pagination
- Identify attributes of interest (nested!)
  - commit
    - committer
      - date
    - tree
      - url
- Visualize and analyze in various ways
  - Commits as events (points) along a timeline
  - Number of commits by any time unit (day, week, month, year) over any period (YTD, year, lifetime of project)
  - Are there any noticeable patterns in commit activity over time?
  - ...

Group activity: Stage 3: code size (advanced)

- For the given repo, pull each commit's git tree based on the commit URLs
  - docs.github.com/en/rest/reference/git#get-a-tree
- Request parameters
  - recursive=true
- Identify attributes of interest (nested!)
  - tree - array with these and other attributes for each element:
    - size
    - path
- Use aggregation over size to compute the ***code size*** for this commit!
  - What unit is it in? Is this an issue?
- Visualize and analyze in various ways
  - Code size over time
  - What are its domain and codomain?
  - Is it partial or total?
  - Is it continuous or discontinuous?
  - Is it monotonic or does it fluctuate?
  - Are there any noticeable patterns in the code size over time?
  - ...

Group activity:
Stage 4: issue density (advanced)

- Combine the issue data from step 1 with the code size data
- Visualize and analyze in various ways
  - Issue density over time
  - Is it monotonic or does it fluctuate?
  - Are there any noticeable patterns in the code size over time?
  - How does it relate to the underlying conceptual material we studied?

Group activity: deliverables

- Brief Scrum-style status updates - Wed 22 Mar and Wed 29 Mar
- Submission - Mon 3 Apr
  - GitHub repo with your MSR scripts and documentation (user and developer-facing)
    - Best to share early on with TA and instructor
  - Sample data and charts for the various stages
  - Answers to the research questions and other relevant insights and reflections
- Presentations and discussion - Wed 5 Apr
- Extra credit
  - Additional repos analyzed
  - Comparisons among analyzed repos/projects
  - Additional research questions you can think of - check with instructor