# Quality Assurance

The role of testing*

Active reviews

*From Prof. Michal Young

# Why Test

- Stupid question?
  - But we need to be clear about goals before we can make reasoned choices regarding the other questions, *who, what, when*, and *how*
  - In general: testing provides the feedback in our "feedback control loop"
- We test to avoid costs
  - Costs during software development
  - Cost of defects in the final product

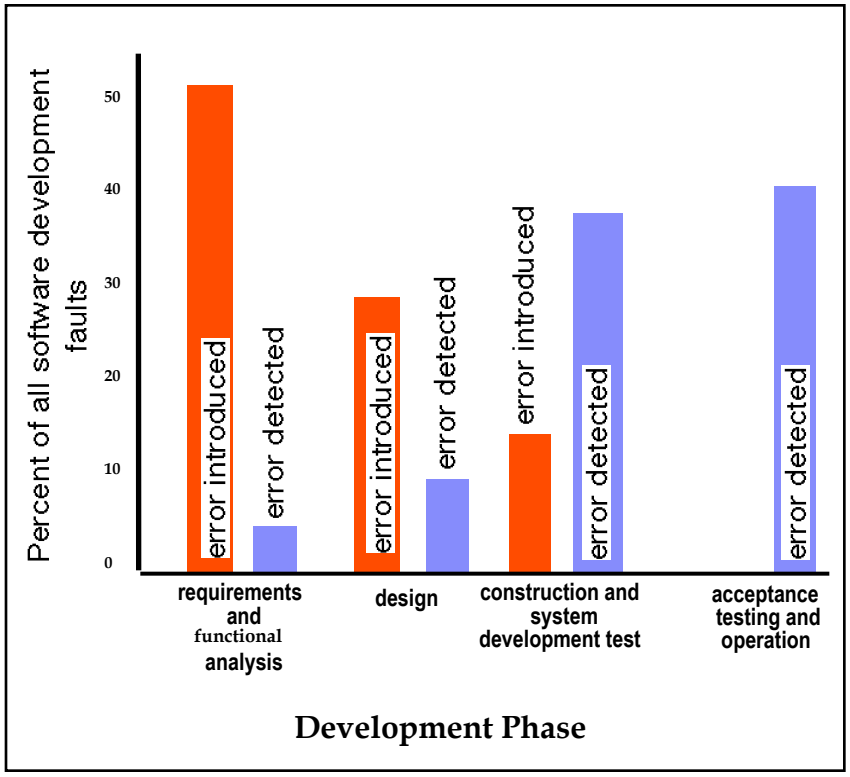
# What about measurement?

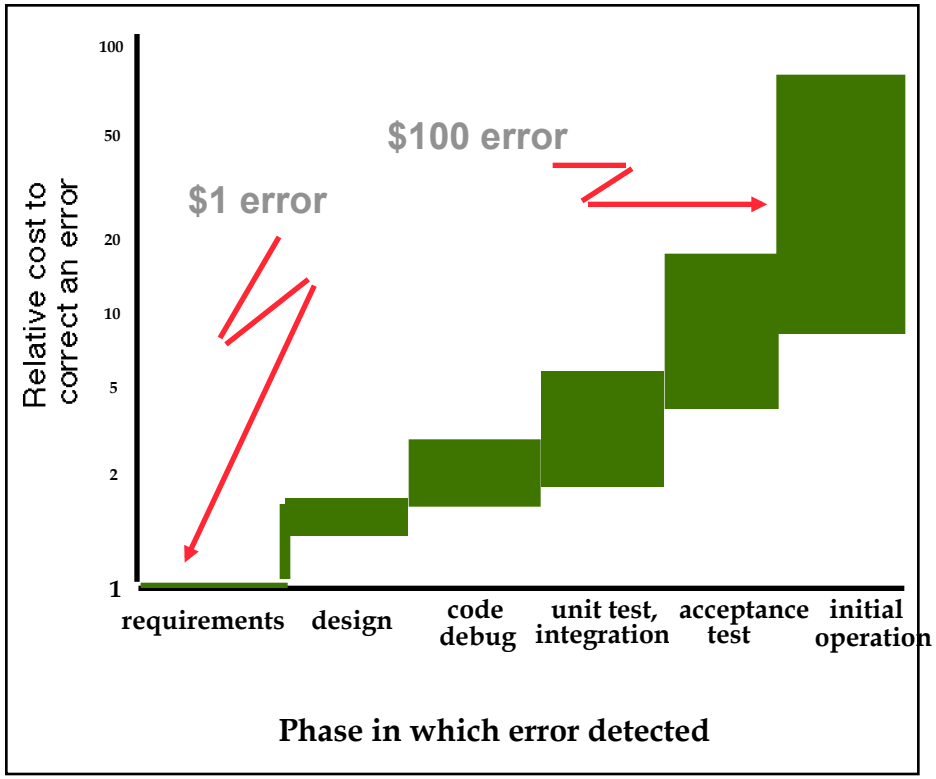
- We can also test to measure the quality of software
  - Is it good enough to release?
  - Should we fix it or replace it?
- Different goals, different approaches
  - (put this aside for the moment ...)

# Importance of Getting Requirements Right

1. The majority of software errors are introduced early in software development

2. The later that software errors are detected, the more costly they are to correct



# Errors, Detection, and Repairs

- Basic observation:
  - Cost of a defect grows *quickly* with time between making an error and fixing it
    - Step function as defects cross scope walls: From programmer to sub-team, from close colleagues to larger team, from module to system, from developers to independent testers and from development to production
- "Early" errors are the most costly
    - Misunderstanding of requirements, architecture that does not support a needed change, ...

# When

- As early as possible
  - Reduce the gap between making an error and fixing it
    - Ideally to “immediately” ... which we call “prevention” or “syntactic checking”
    - E.g., error detection/correction in Eclipse, other programming environments
- Throughout development
  - People make mistakes in every activity, so every work product must be tested as soon as possible

# What

- Narrow view:
  - Testing is executing a program and comparing actual results to expected results
- Wider view:
  - “Testing” is shorthand for a variety of activities: anything we can do to check for defects
  - Dynamic program testing is the most common activity when the artifact is program code
  - Also, reviews, analysis of models, automated checks; we usually need several

# Choosing What

- For every work product, we ask: How can I find defects as early as possible
  - Ex: How can I find defects in software architecture before we've designed all the modules? How can I find defects in my module code before it's integrated into the system?
- Divide and conquer
  - What properties can be checked automatically?
  - What properties can be (effectively) tested dynamically?
  - How can I make reviews cost-effective?

# Verification and Validation: Divide and Conquer

- Validation vs. Verification
  - Are we building the right product? vs. Are we building it right?
  - Crossing from judgment to precise, checkable correctness property. Verification is at least partly automatable, validation is not
- Correctness is a *relation* between spec and implementation
  - To make a property verifiable (testable, checkable, ...) we must capture the property in a spec

# Divide and Conquer: Usability

- Real requirement:
  - The product must be usable. Users with characteristics XXX should learn to use it effectively within 30 minutes, and should thereafter complete task T within S seconds with error rate E.
  - Hard and expensive (but important) to test. We probably can't test it after every trivial change to the product.
- Divide and conquer:
  - Validate the user interface design.
  - Verify the user interface implementation: Is it consistent with the design? Does it violate any of the (precisely stated) requirements?

# Testing: Who (from Why and When)

- Different goals, different actors
  - Usability example:  Validating the UI design is an expensive activity requiring expertise in usability, best carried out in a usability lab.
  - Verifying compliance to UI design should be simple and mostly automatable, carried out by developers and development testers.
  - Measuring reliability is yet another goal, best carried out by an independent team.
- When: Now
  - Timeliness often constrains responsibility, e.g., unit testing is carried out by developers (but perhaps reviewed by others)

# Who

- Cost of a defect rises dramatically at architectural and sub-team boundaries
  - It's cheap for me to fix the bug I just created in my module. It's much, much more expensive to find, understand, and fix a bug in a module made by my teammate who is sleeping 3000 miles away.

=> Test cases are part of good module interface designs

=> Module tests should be thorough and completed before a module (or revision) becomes part of the baseline used by others

# The Long When

- Test execution is just one part of testing
  - And it needs to be a very cheap, automated part, because we should re-test the program over and over as it evolves
- Test design can often be done much earlier
  - Can begin building tests based on use cases and other requirements
  - Part of a good system design is devising acceptance test cases
- Test design is also a test of specifications
  - Is this specification precise, or ambiguous? Can I effectively check whether an implementation satisfies it?

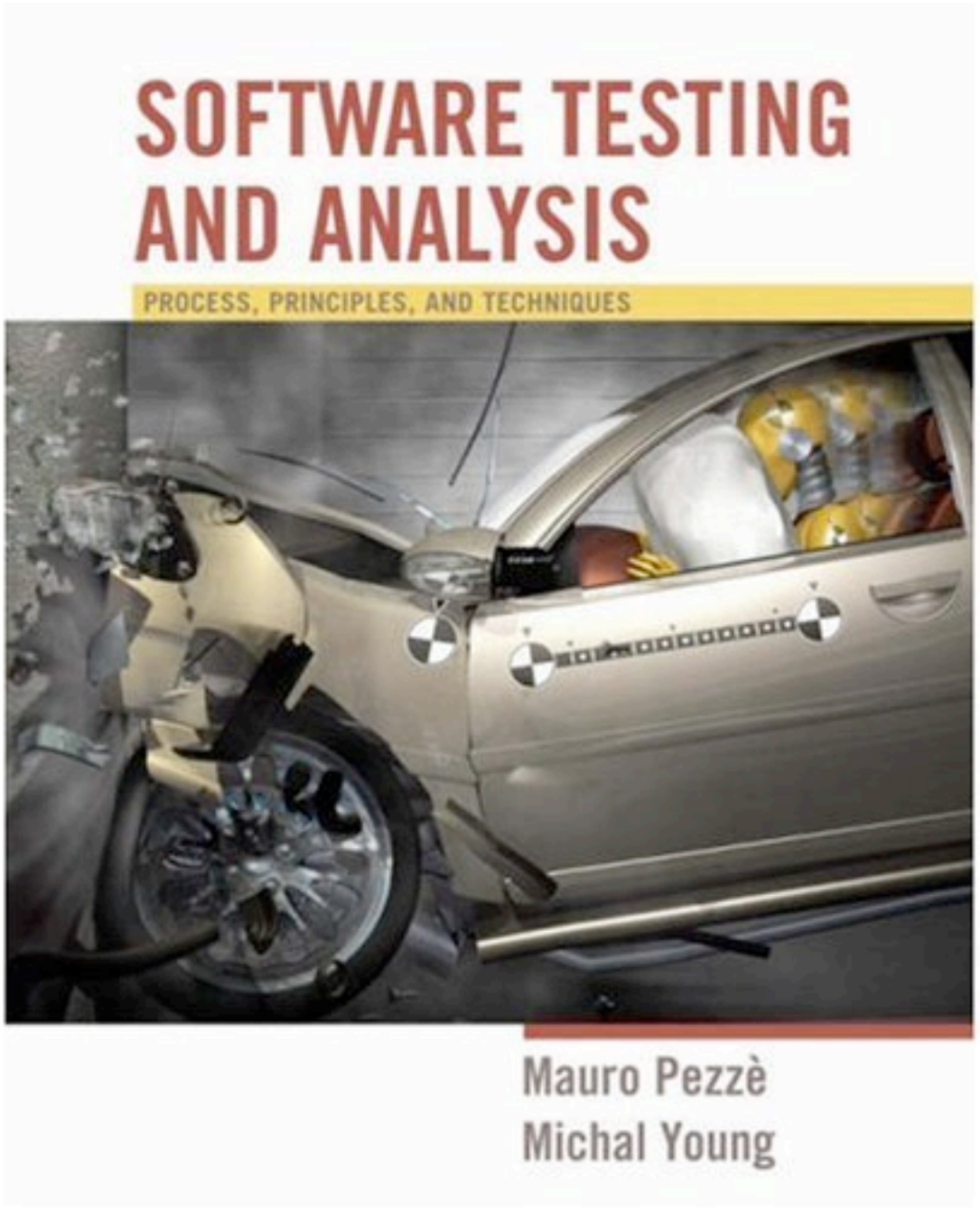
# How (from why, who, when, what)

- Black box: Test design is part of designing good specifications
  - This will change specs, in a good way. Factoring validation from verification is particularly hard, but particularly cost-effective as it leverages and focuses expensive human judgment
- White (or glass) box: Test design from program design
  - Executing every statement or branch does not guarantee good tests, but omitting a statement is a bad smell.

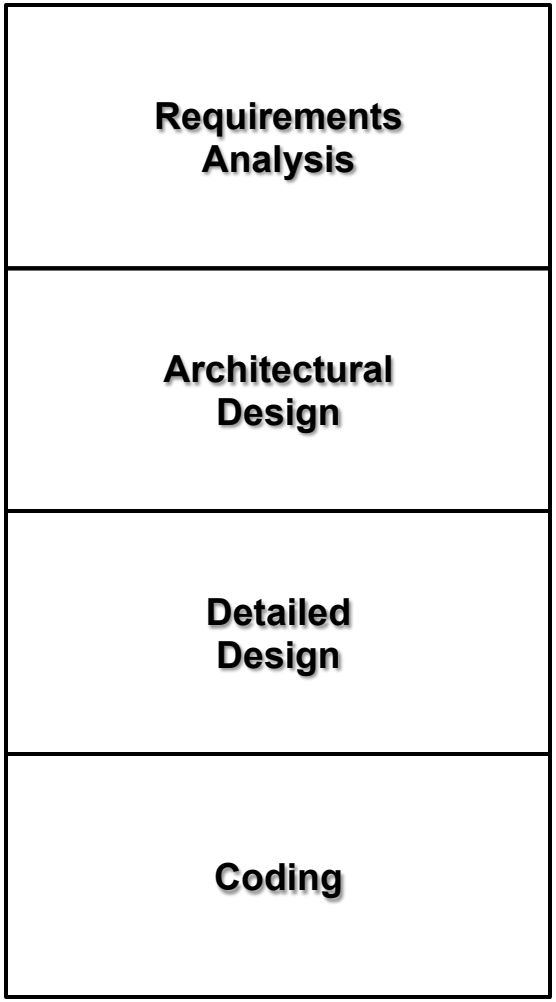
# More!

- Testing and process refinement:
  - Effective organizations track and classify defects. What do we typically get wrong? Which bugs cost us most? What causes them? How can we cost-effectively prevent or detect them?
- Testing and incentive structures:
  - It's easy to accidentally create perverse incentives, e.g., to turn in modules on schedule with latent bugs. If the incentive structure is broken, nothing else will work.

# Plug


SOFTWARE TESTING
AND ANALYSIS
PROCESS, PRINCIPLES, AND TECHNIQUES
Mauro Pezzè
Michal Young

# Summary: Quality is Cumulative




- Are the requirements valid?
- Complete? Consistent? Implementable?
- Testable?

- Does the design satisfy requirements?
- Are all functional capabilities included?
- Are qualities addressed (performance, maintainability, usability, etc.?

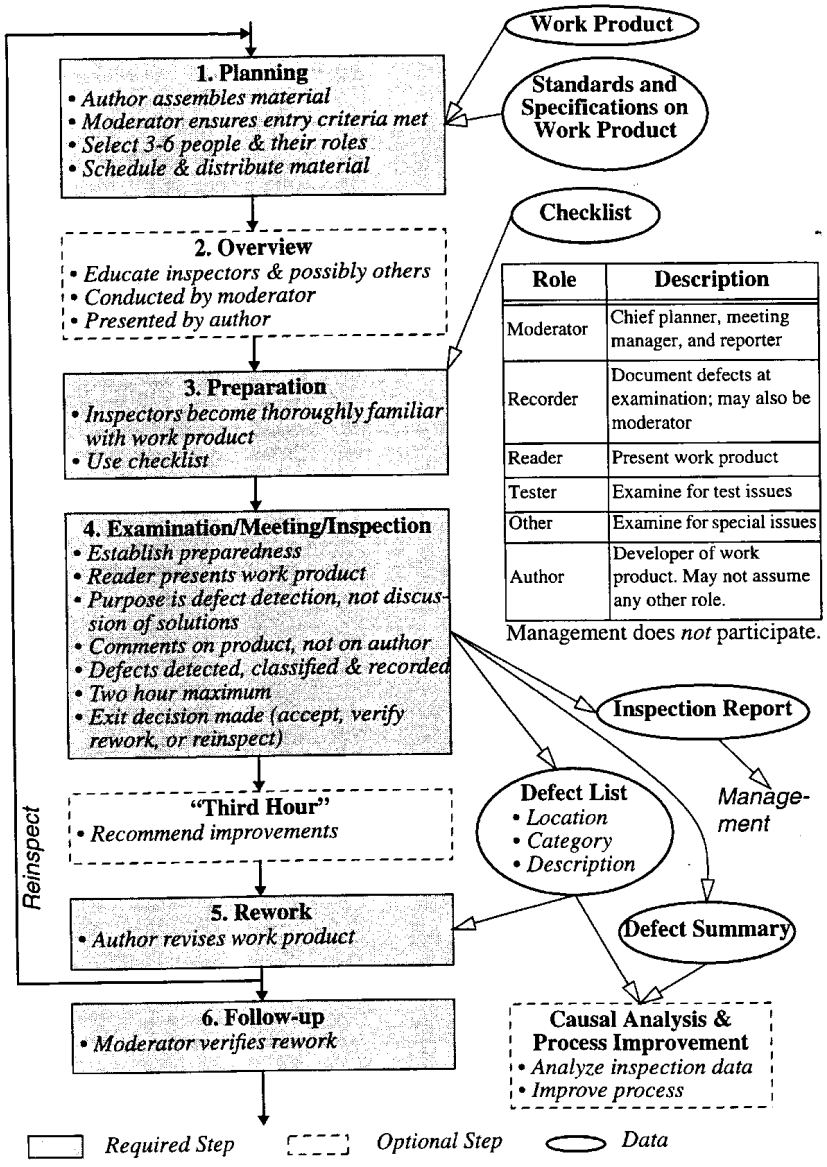
- Do the modules work together to implement all the functionality?
- Are likely changes encapsulated?
- Is every module well defined

- Implement the required functionality?
- Race conditions? Memory leaks? Buffer overflow?

# Active Reviews

# Peer Reviews

- Peer Review: a process by which a *software product is examined by peers of the product's authors with the goal of finding defects*
- Why do we do peer reviews?
  - Review is often the only available verification method before code exists
  - Formal peer reviews (inspections) instill some discipline in the review process
- Particularly important for distributed teams
  - Supports communication and visibility
  - Provides feedback on both *quality and understanding*
    - i.e., makes the communication effectiveness and level of understanding visible
  - A good review shows communication is working!




| Role | Description |
|---|---|
| Moderator | Chief planner, meeting manager, and reporter |
| Recorder | Document defects at examination; may also be moderator |
| Reader | Present work product |
| Tester | Examine for test issues |
| Other | Examine for special issues |
| Author | Developer of work product. May not assume any other role. |

Management does *not* participate.

Figure 1. The software inspection process (IEEE Std 1028).



Example: IEEE software inspection process (aka Fagan Inspection)

# Effectiveness of Peer Reviews

- Generally considered *most effective manual technique for detecting defects*
  - Analysis of 12,000 development projects showed defect detection rate of 60-65% for formal inspection 30% for testing
  - Bell-Northern found 1 hour code inspecting saves 2 to 4 hours code testing
  - Effect is magnified in earlier inspections (e.g., 30 times for requirements in one study)
- Means that you should be doing peer reviews, but…
  - Doesn't mean that manual inspections cannot be improved
  - Doesn't mean that manual inspections are the best way to check for every properties (e.g., completeness)
    - Should be one component of the overall V&V process

# Peer Review Problems

- Tendency for reviews to be incomplete and shallow
- Reviewers typically swamped with information, much of it irrelevant to the review purpose
- Reviewers lack clear individual responsibility
- Effectiveness depends on reviewers to initiate actions
  - Review process requires reviewers to speak out
  - Keeping quiet gives lowest personal risk
  - Rewards of finding errors are unclear at best

# Peer Review Problems (2)

- Large meeting size hampers effectiveness, increases cost
  - Makes detailed discussion difficult
  - Few present reviewers have interest/expertise on any one issue
  - Wastes everyone else's time and energy
- No way to cross-check unstated assumptions

# Qualities of Effective Review

- Ensures adequate coverage of artifact in breadth and depth
- Reviewers review only issues on which they have expertise
- Review process is active: i.e., performing the review produces visible output (risk in in doing nothing)
- Individual responsibilities are clear and fulfilling them is evidence of a job well done.

# Qualities of Effective Review (2)

- Review process focuses on finding specific kinds of errors.
- Limit meetings to focused groups and purposes requiring common understanding or synergy
  - Permit detailed discussion of issues
  - Expose where assumptions differ

# Active Reviews

- Goal: Make the reviewer(s) think hard about what they are reviewing

1) Identify several types of review each targeting a different type of error (e.g., UI behavior, consistency between safety assertions and functions).
2) Identify appropriate classes of reviewers for each type of review (specialists, potential users, methodology experts)
3) Assign reviews to achieve coverage: each applicable type of review is applied to each part of the specification

# Active Reviews (2)

4) Design review questionnaires (key difference)
   - Define questions that the review must answer by using the specification
   - Target questions to bring out key issues
   - Phrase questions to require "active" answers (not just "yes")
5) Review consists of filling out questionnaires defining
   - Section to be reviewed
   - Properties the review should check
   - Questions the reviewer must answer
6) Review process: overview, review, meet
   - One-on-one or small, similar group
   - Focus on discussion of issues identified in review
   - Purpose of discussion is understanding of the issue (not necessarily agreement)

# Examples

- In practice: an active review asks a qualified reviewer to check a specific part of a work product for specific kinds of defects by answering specific questions, e.g.,
  - Ask a designer to check the functional completeness by showing the calls sequences sufficient to implement a set of use cases
  - Ask a systems analyst to check the ability to create required subsets by showing which modules would use which
  - As a developer to check the data validity of a module's specification by showing what the output would be for in-range and out-of-range values
  - Ask a technical writer to check the SRS for grammatical errors
- Can be applied to any kind of artifact from requirements to code

# Conventional vs. Active Questions

- **Goal: Make the reviewer(s) think hard about what they are reviewing***
  - **Define questions that the review must answer by using the specification**
  - **Target questions to bring out key issues**
  - **Phrase questions to require "active" answers (not just "yes")**

| | |
|---|---|
| Are exceptions defined for every program? | For each access program in the module, what exceptions that can occur? |
| Are the right exceptions defined for every program? | What is the the range or set of legal values? |
| Are the data types defined? | For each data type, what are • an expression for a literal value of that data type; • a declaration statement to declare a variable for that type; • the greatest and least values in the range of that data type? |
| Are the programs sufficient? | Write a short pseudo-code program that uses the design to accomplish {some defined task}. |

# Role of Use Cases

- Use cases or scenarios can be effectively used in active review
- Apply requirements scenarios to verify design against requirements
  - "Show the sequence of program calls that would implement use case C"
  - "Which modules would have to change to add feature F (a likely change)?"
- Conversely, can check properties ask the reviewer to construct scenarios
  - "What sequence of calls would result in an exception E?"

# Why Active Reviews Work

- Focuses reviewer's skills and energies where they have skills and where those skills are needed
  - Questionnaire allows reviewers to concentrate on one concern at a time
  - No one wastes time on parts of the document where there is little possibility of return.
- Largest part of review process (filling out questionnaires) is conducted independently and in parallel
- Reviewers must participate actively but need not risk speaking out in large meetings
- Downside: much more work for V&V (but can be productively pursued in parallel with document creation)

# Summary

- Need to do reviews to find defects
- Active reviews are more efficient and effective but may take more effort

# Questions?

# Schedule

- Thanksgiving Break
- Week 10
  - Tue: course review
  - Thur: project presentations (Deschutes?)
    - Will post presentation guidelines
    - About 20 minutes for demo and lessons learned
    - Can we take longer for class?
- Final:
  - Scheduled Fri., Dec, 9th, 8:00
  - Alternate date?