FACTS AND FALLACIES OF SOFTWARE ENGINEERING

By Robert L. Glass.

Publisher: Addison Wesley. Pub Date: October 25, 2002. ISBN: 0-321-11742-5.

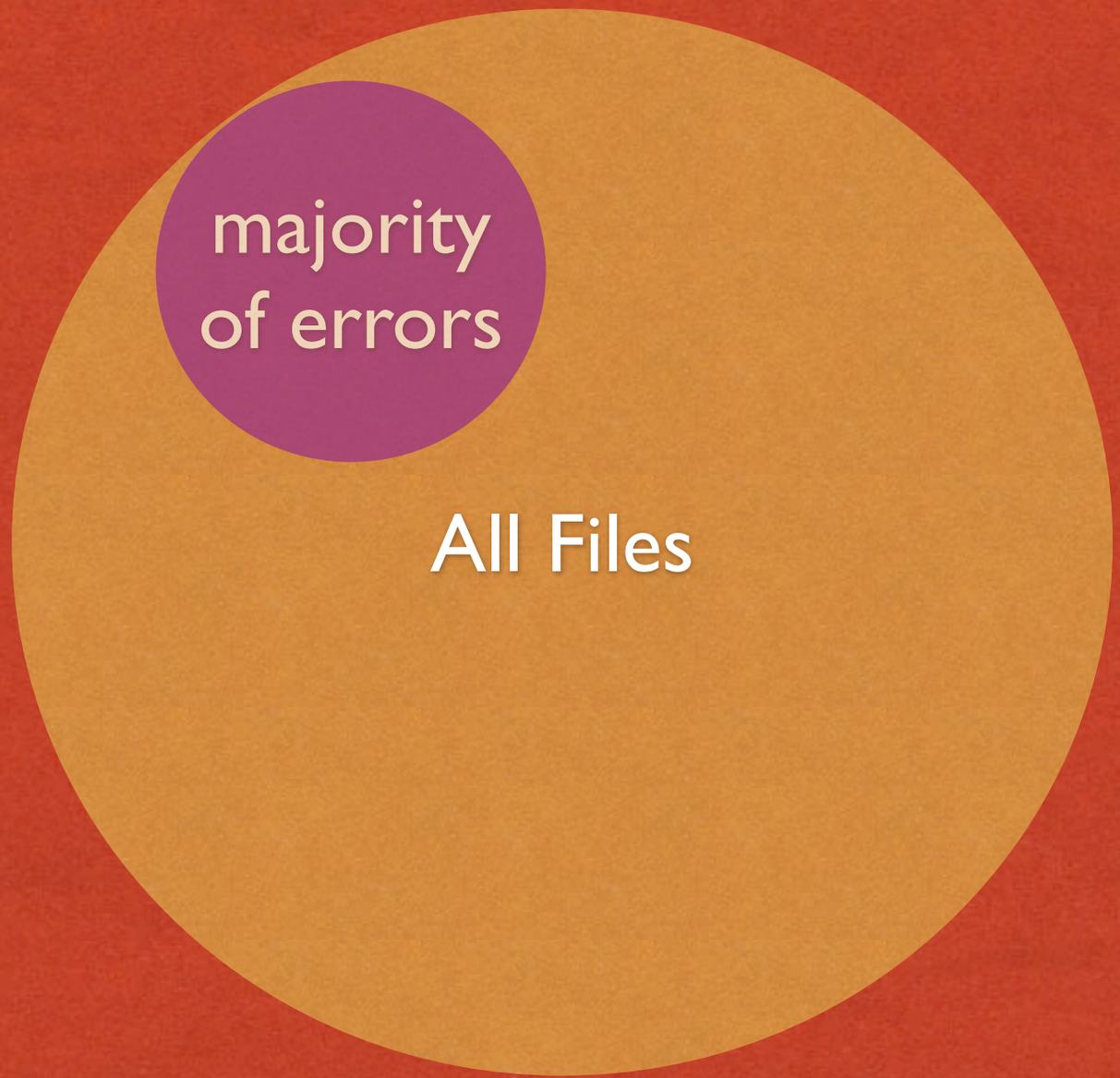
FACT 49.
Errors tend to cluster.

Cindee Madison
UC Berkeley

ERRORS TEND TO CLUSTER

- small number of files have large number of errors

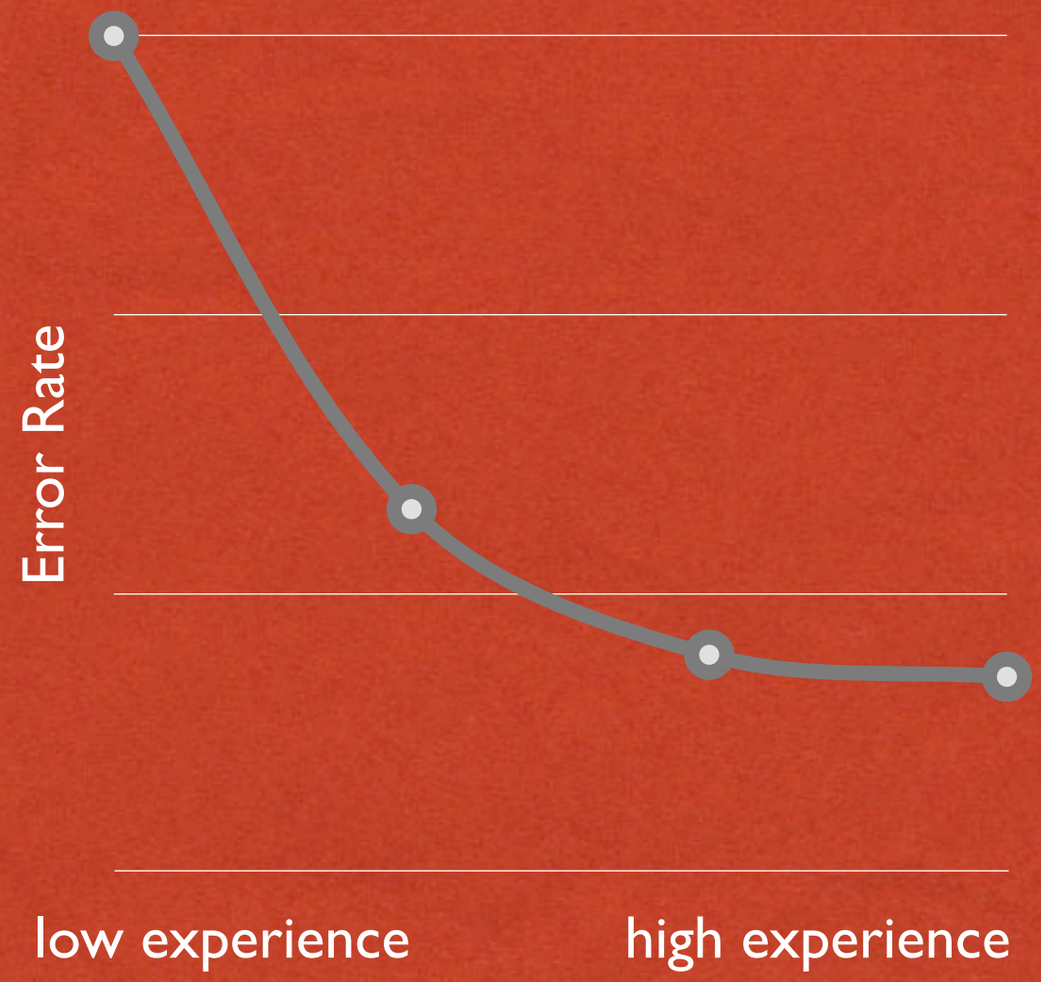
- majority of files have 0-1 error



WHY DO ERRORS CLUSTER?

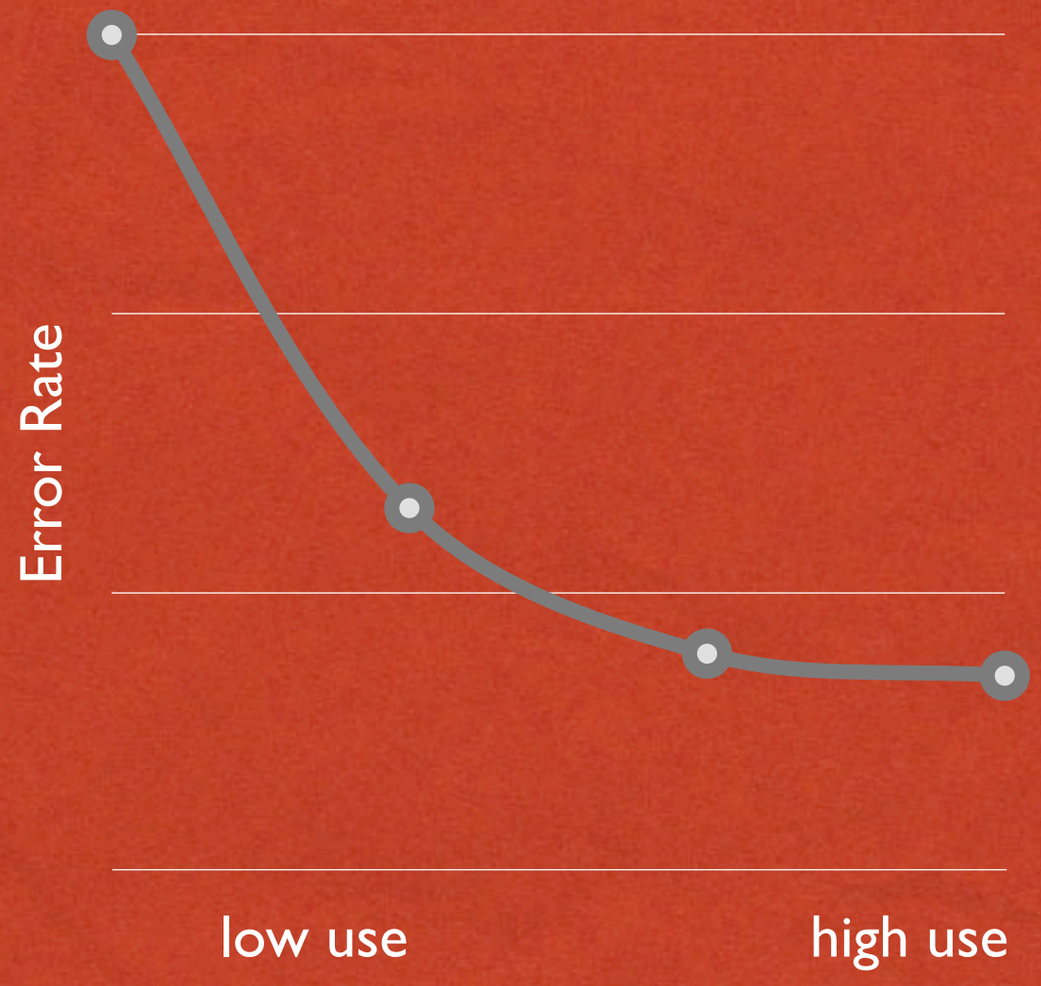
Programmer inexperience with domain specific logic

Impact of Domain Specific Knowledge



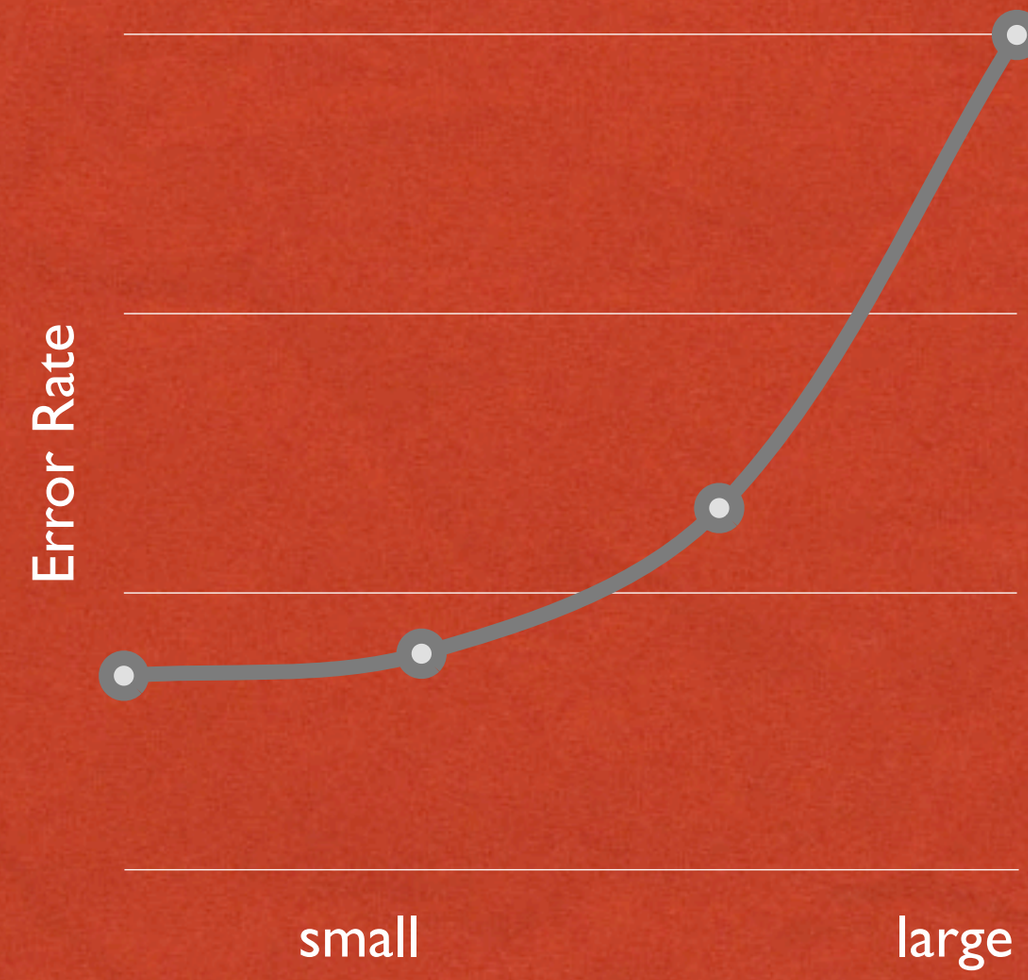
An Empirical Study of Operating Systems Errors 2000 A Chou, J Yang, B Chelf, S Hallem, and D Engler
Computer Systems Laboratory Stanford University

Heavily used code has fewer errors



An Empirical Study of Operating Systems Errors 2000 A Chou, J Yang, B Chelf, S Hallem, and D Engler
Computer Systems Laboratory Stanford University

As functions grow larger the error rate increases

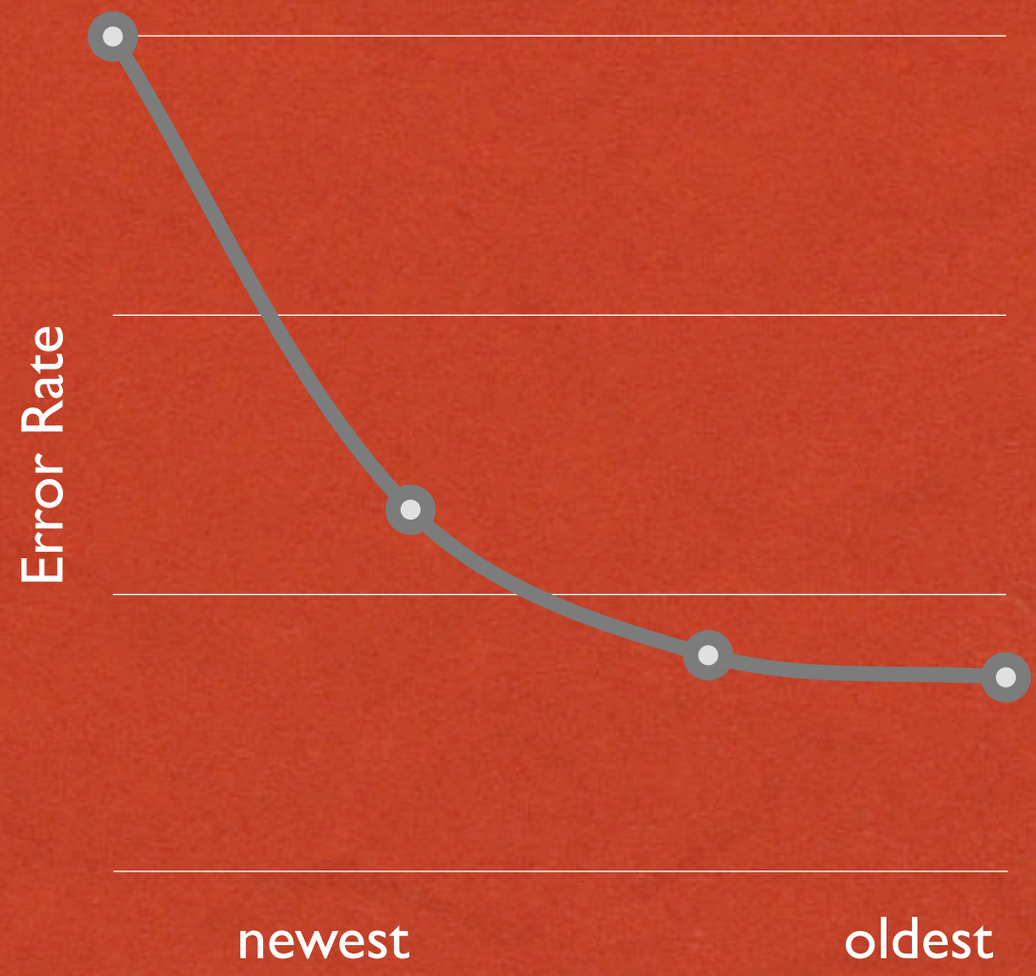


An Empirical Study of Operating Systems Errors 2000 A Chou, J Yang, B Chelf, S Hallem, and D Engler
Computer Systems Laboratory Stanford University

"Code Hardening"

Older files tend to have fewer errors "code hardening"

newest quartile of files has 2X errors of oldest quartile



An Empirical Study of Operating Systems Errors 2000 A Chou, J Yang, B Chelf, S Hallem, and D Engler
Computer Systems Laboratory Stanford University

Primary Cause of Errors

- ignorance of interface / system rules combined with “copy and paste”
- “34 of the errors were caused by cut-and-paste: one of the errors was copied in 10 places and another in 24”

An Empirical Study of Operating Systems Errors 2000 A Chou, J Yang, B Chelf, S Hallem, and D Engler
Computer Systems Laboratory Stanford University

Code Reuse Caveat

Older files have fewer errors,
However .....

- repurposing modules resulted in increased time and effort to correct errors
- Cost of developing a new implementation is less than the cost of adapting modules to a new specification

V. Basili and B. Perricone, Software Errors and Complexity: An Empirical Investigation, Communications of the ACM, vol. 27(1): 42-52, January 1984

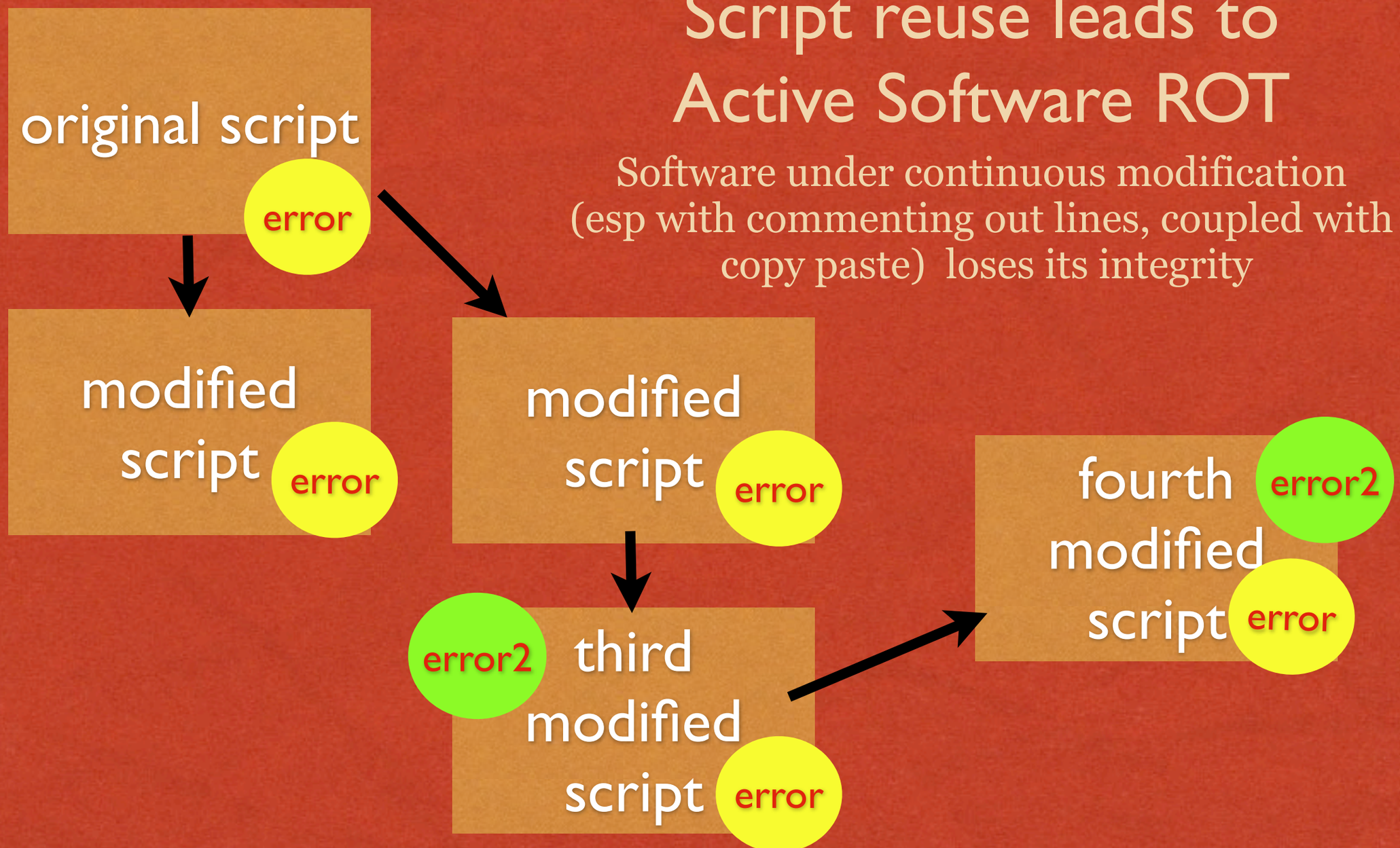
DESIGNING TO REDUCE ERRORS

- if you find more than one error, keep looking, there are likely more
- keep code modular, break complex logic into smaller parts
- increase reuse ***without*** modification
- heavily test difficult logic, or areas of weak specification understanding
- consider cost of modifying code for new specification, vs designing a new implementation

Example

Script reuse leads to Active Software ROT

Software under continuous modification (esp with commenting out lines, coupled with copy paste) loses its integrity



Example (solution?)

- create libraries of common generic logic and algorithms
  - well tested, and reviewed
  - errors fixed in library benefits all child scripts
- write new wrappers for specific implementations from scratch
- bug tracking tools (to keep all users aware of changes to library)