The CESM Automated Test System

Unlocking the door to more efficient, robust CESM development

Bill Sacks,
Jay Shollenberger,
and other CSEG members

This talk will be posted at:
http://www2.cgd.ucar.edu/sections/cseg/tutorials

Welcome to the first of what we hope will become a long series of CSEG coffee talks. CSEG – the CESM software engineering group – plans to lead one tutorial or discussion like this every few months. These will be aimed at all developers of CESM – primarily the scientists who develop the model. We'll assume you have some basic knowledge of CESM, such as how to create and run a case. But we'll try not to assume any advanced knowledge of CESM development. We really intend for these to be accessible to people just getting started with CESM.

Please give us feedback on what you think about this, and especially what topics you would like us to cover.

Today's talk will be on the CESM automated test system.

Although I'm presenting this, others – particularly Jay – get the credit for putting these tools together

Outline

- Intro & motivation
- Basics of using the automated test system
- Comparing against baselines
- Running a whole test suite
- Summary
- Appendix: References for later use

Outline

- Intro & motivation
- Basics of using the automated test system
- Comparing against baselines
- Running a whole test suite
- Summary
- Appendix: References for later use

Life Before Automated Testing

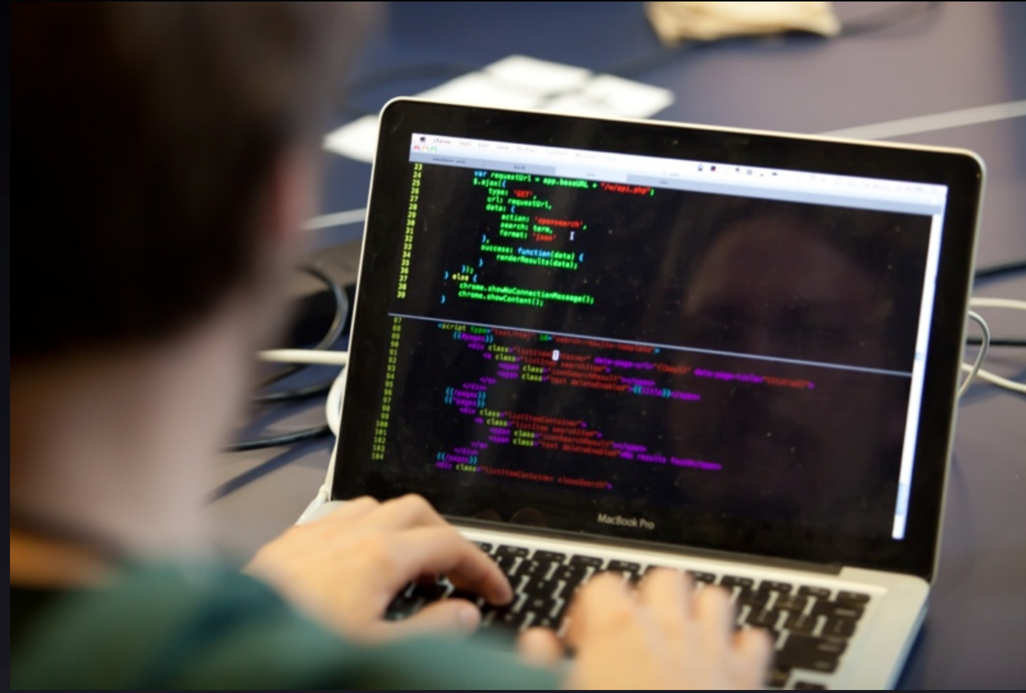


Image credits:
http://commons.wikimedia.org/wiki/File:Typing_computer_screen_reflection.jpg
http://en.wikipedia.org/wiki/Crossed_fingers
http://revaustinmiles.com/index.php/more/435

Life Before Automated Testing

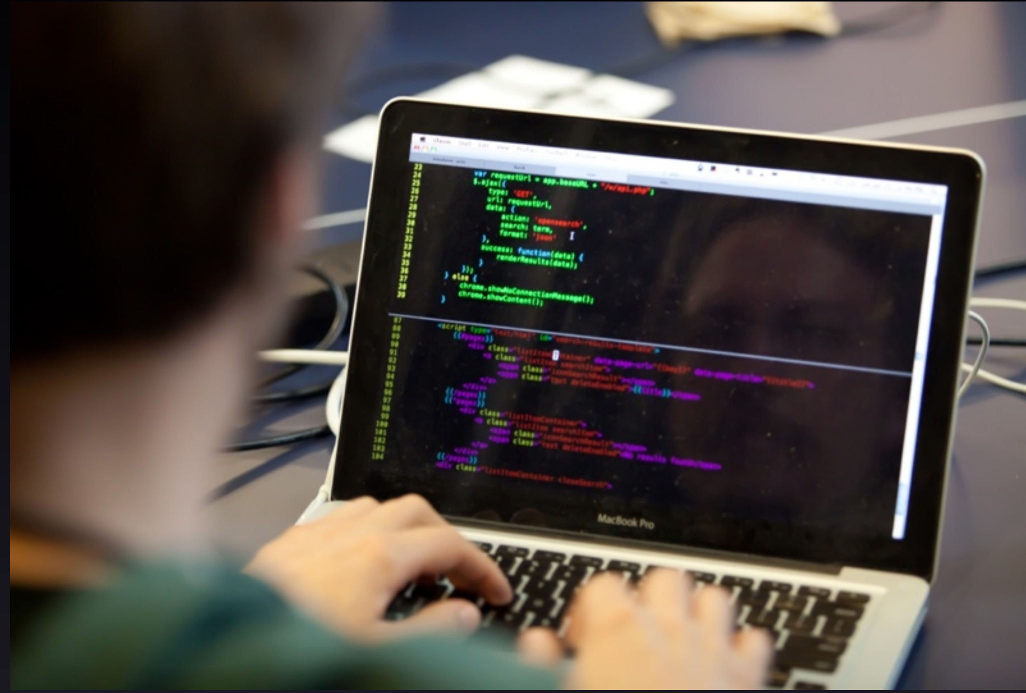
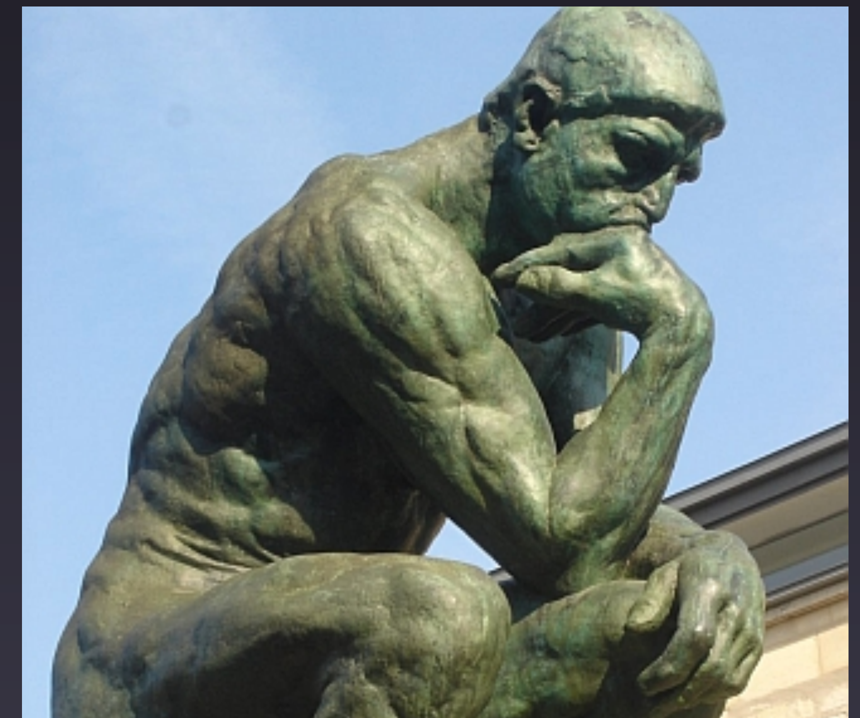


Image credits:
http://commons.wikimedia.org/wiki/File:Typing_computer_screen_reflection.jpg
http://en.wikipedia.org/wiki/Crossed_fingers
http://revaustinmiles.com/index.php/more/435

Life Before Automated Testing



Friday, March 14, 14

Image credits:
http://commons.wikimedia.org/wiki/File:Typing_computer_screen_reflection.jpg
http://en.wikipedia.org/wiki/Crossed_fingers
http://revaustinmiles.com/index.php/more/435

What Do We Want to Test?

Functionality Tests

- Runs to completion
- Restarts bit-for-bit
- Results independent of processor count
- Threading
- Compilation with debug flags, e.g., to pick up:
  - array bounds problems
  - floating point errors
- And other specialty tests

Independent of processor count: for SOME components.

Threading: important for performance

Rationale for some of this is reproducibility: If you want to redo a run, or compare an experiment vs control, you don't want to specify: "Okay, you need to run with exactly 1,245 processors, and restart the model every 2 years... except for the first 10 years of the simulation, actually restart every year"

So this is a way of ensuring that you don't have to specify all those details.

But also, these tests often pick up more fundamental bugs – e.g., processor count not bfb because there is a whole-array assignment where there should be assignment just to one element of the array

What Do We Want to Test?

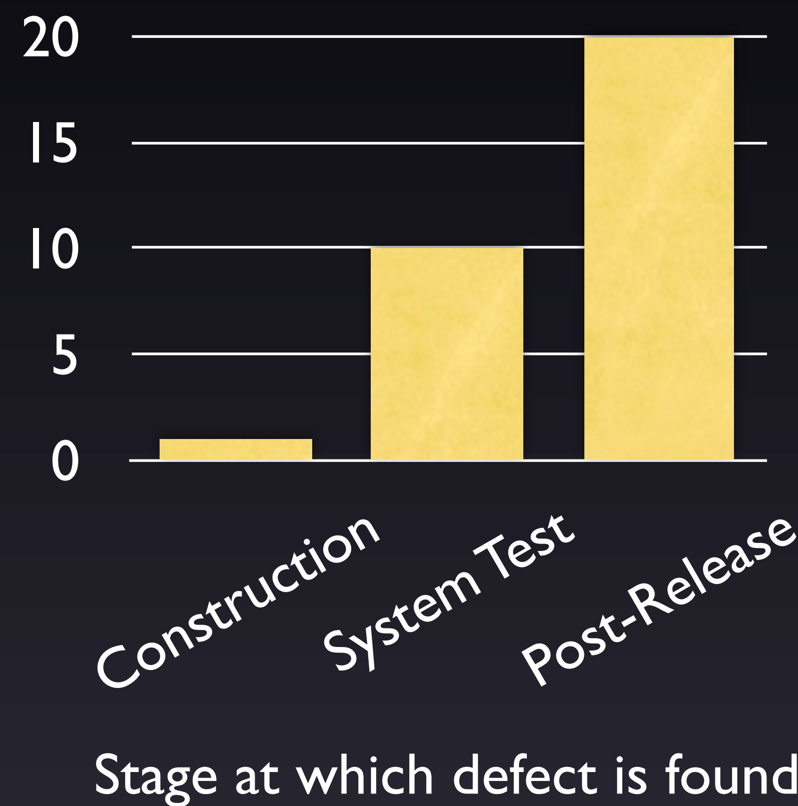
"I didn't break any other functionality"

- Make sure other model configurations still work
  - Example: Making sure CLM still works when you turn on prognostic crops
- Make sure code works with other compilers
- If you expect a set of changes to maintain identical answers, make sure that's true
  - Terminology: "Bit-for-bit"

This is important for your own science, too: Even though you may just be developing one little corner of the model, at the end of the day, you're still running the whole model, so you want to be sure that the other pieces of the model still work the same as before.

All of this is a lot to remember. And that's the motivation for the automated test suite: Then you don't have to remember to test all of these different things. In fact, you don't even have to know what threading is. The automated test suite will test these things for you and tell you if there is a problem.

Isn’t Testing the Responsibility of Software Engineers?



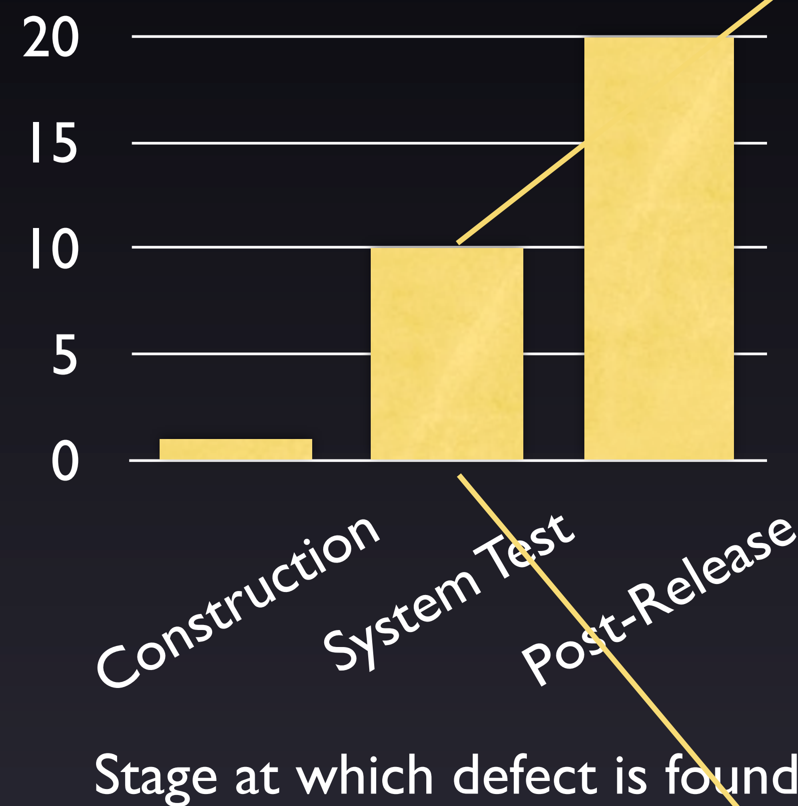
Steve McConnell (2004), Code Complete
2nd edition, p. 29

Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

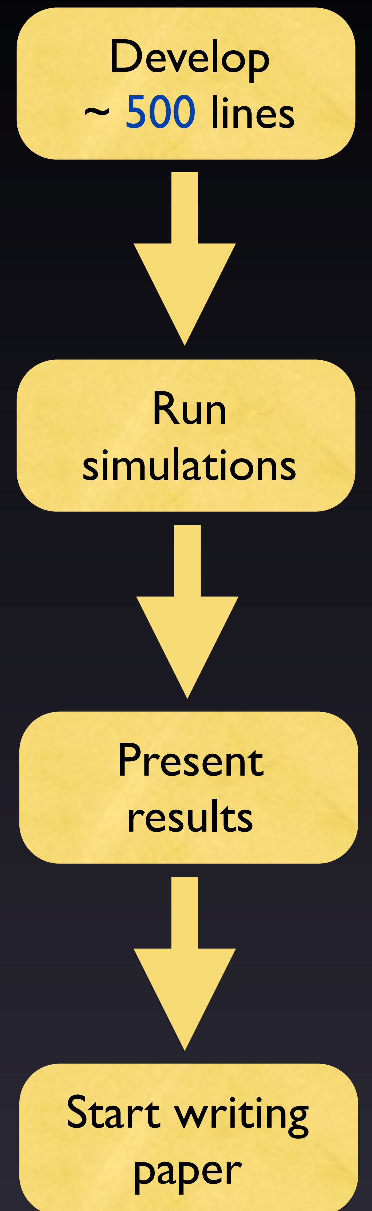
Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

Left-hand flow chart: time frame of cycle ~ days

Isn't Testing the Responsibility of Software Engineers?



Steve McConnell (2004), Code Complete 2nd edition, p. 29

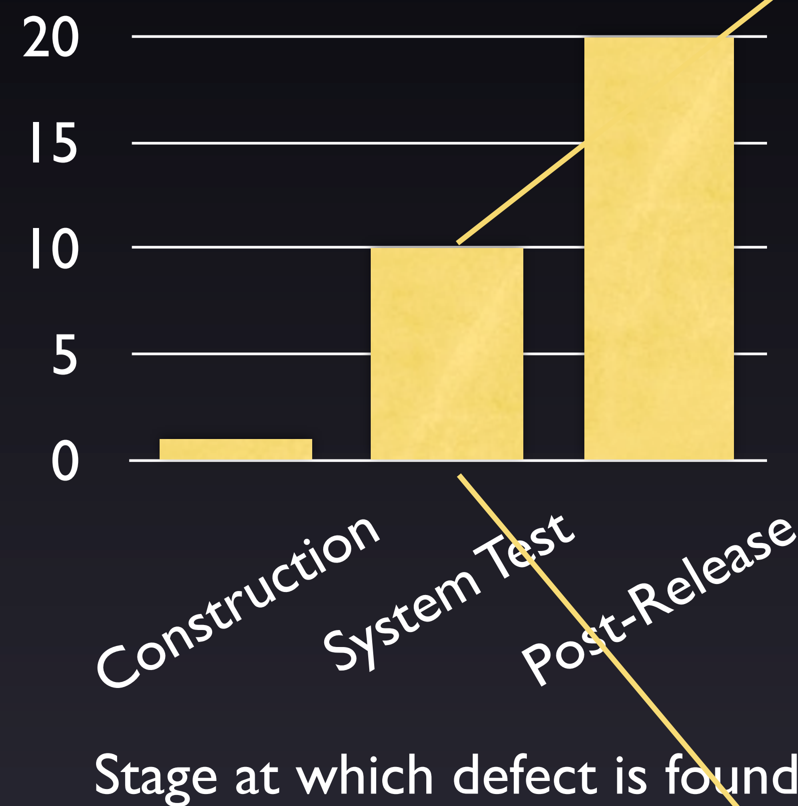


Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

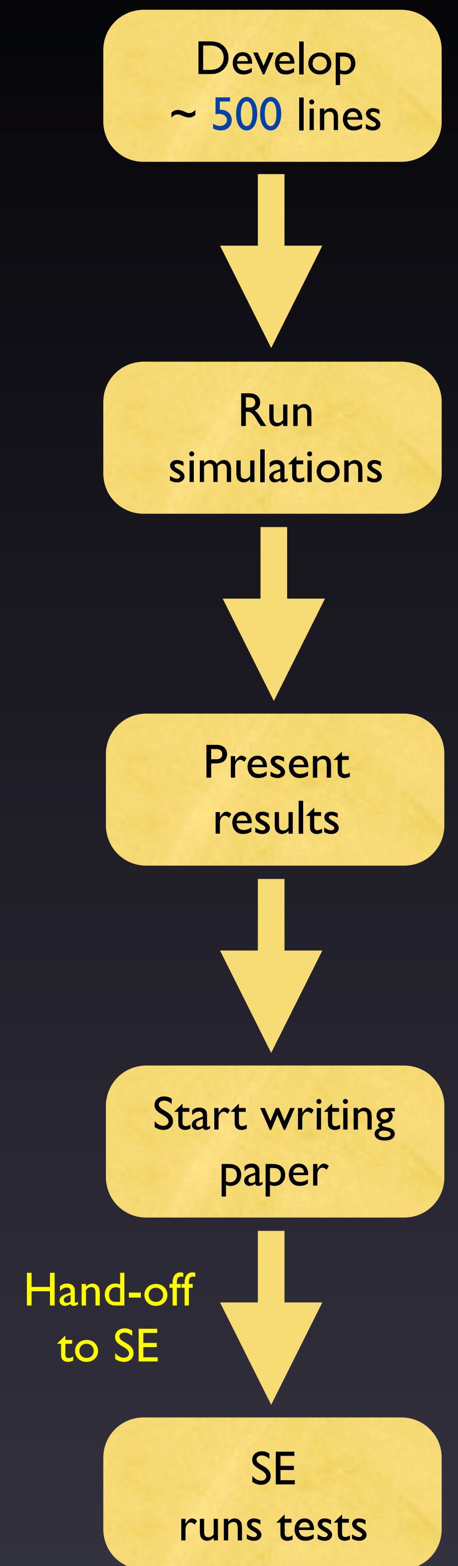
Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

Left-hand flow chart: time frame of cycle ~ days

Isn't Testing the Responsibility of Software Engineers?



Steve McConnell (2004), Code Complete
2nd edition, p. 29

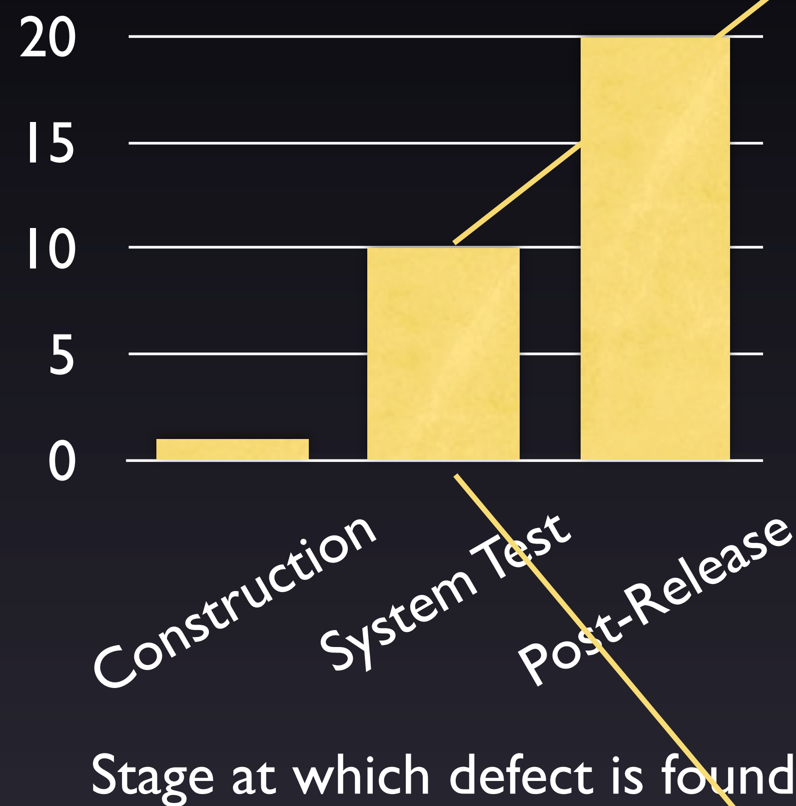


Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

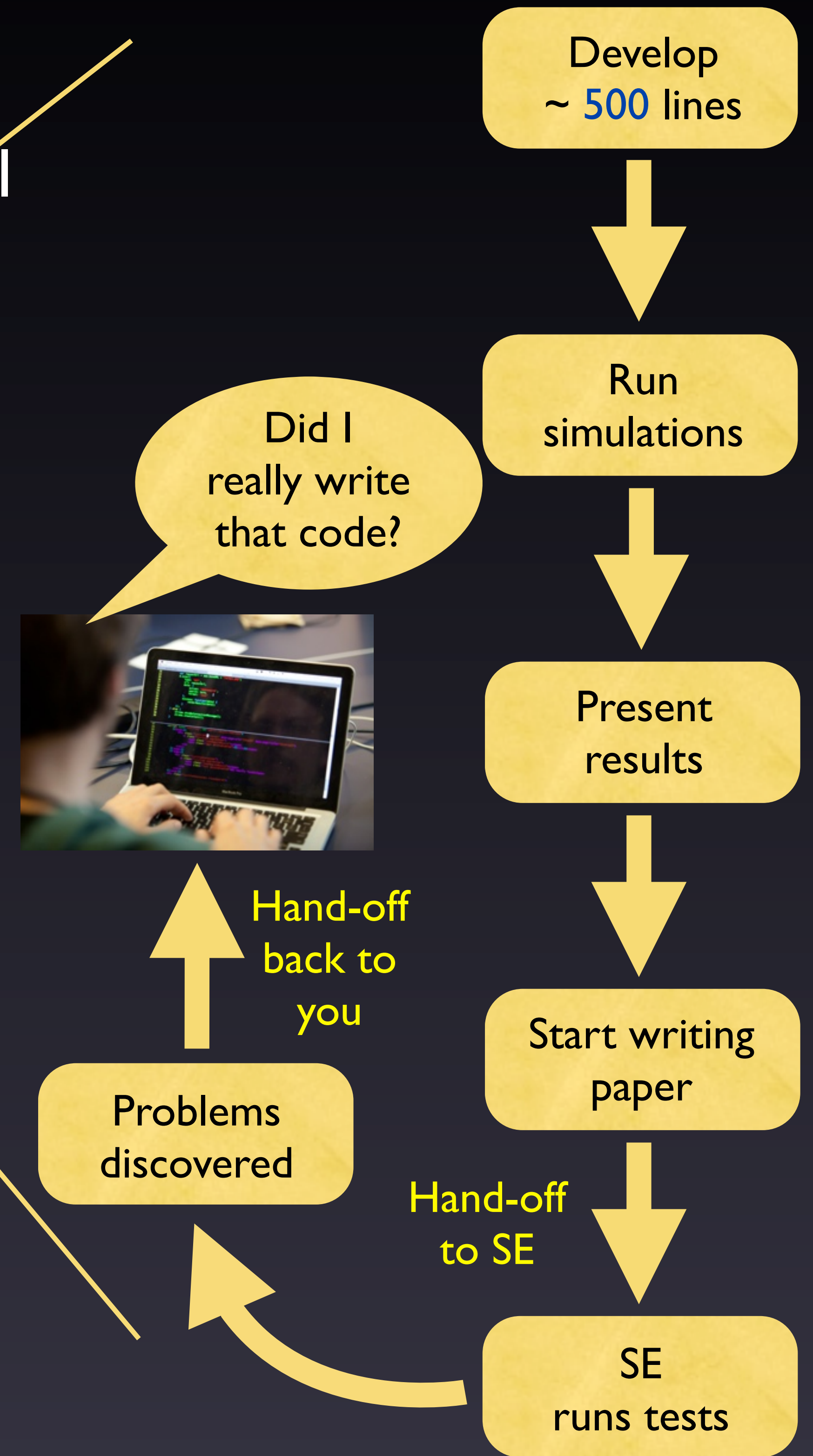
Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

Left-hand flow chart: time frame of cycle ~ days

Isn't Testing the Responsibility of Software Engineers?



Steve McConnell (2004), Code Complete 2nd edition, p. 29

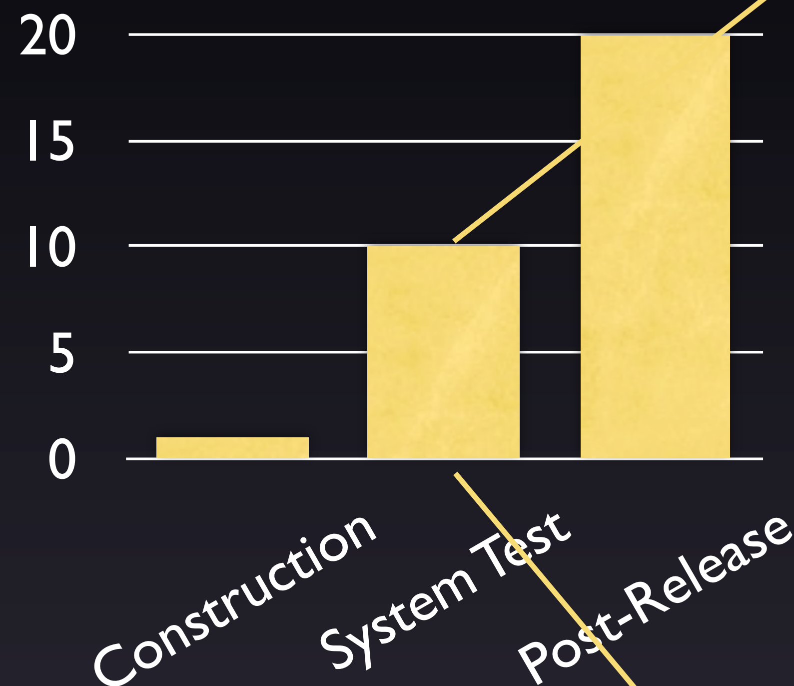


Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

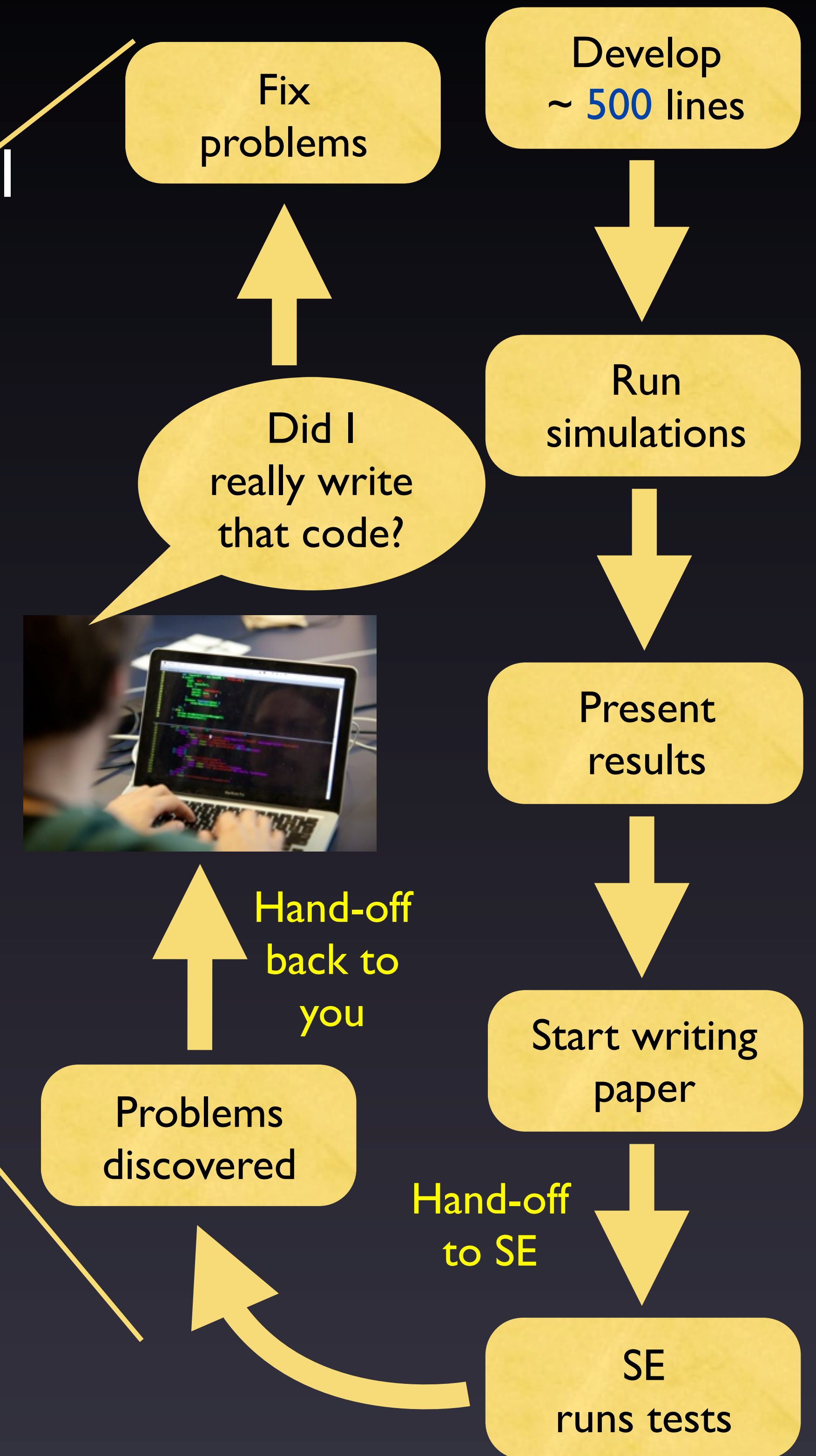
Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

Left-hand flow chart: time frame of cycle ~ days

Isn't Testing the Responsibility of Software Engineers?



Steve McConnell (2004), Code Complete
2nd edition, p. 29

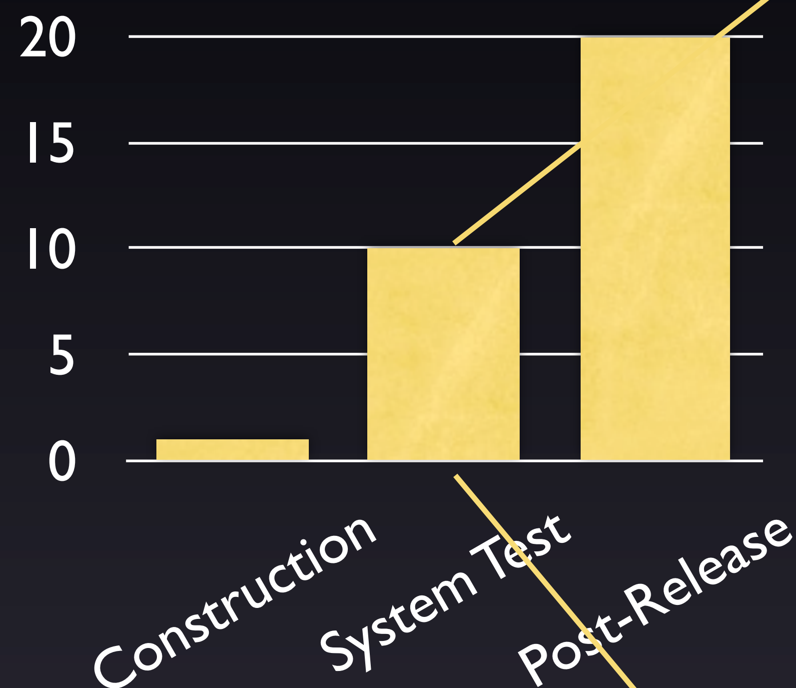


Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

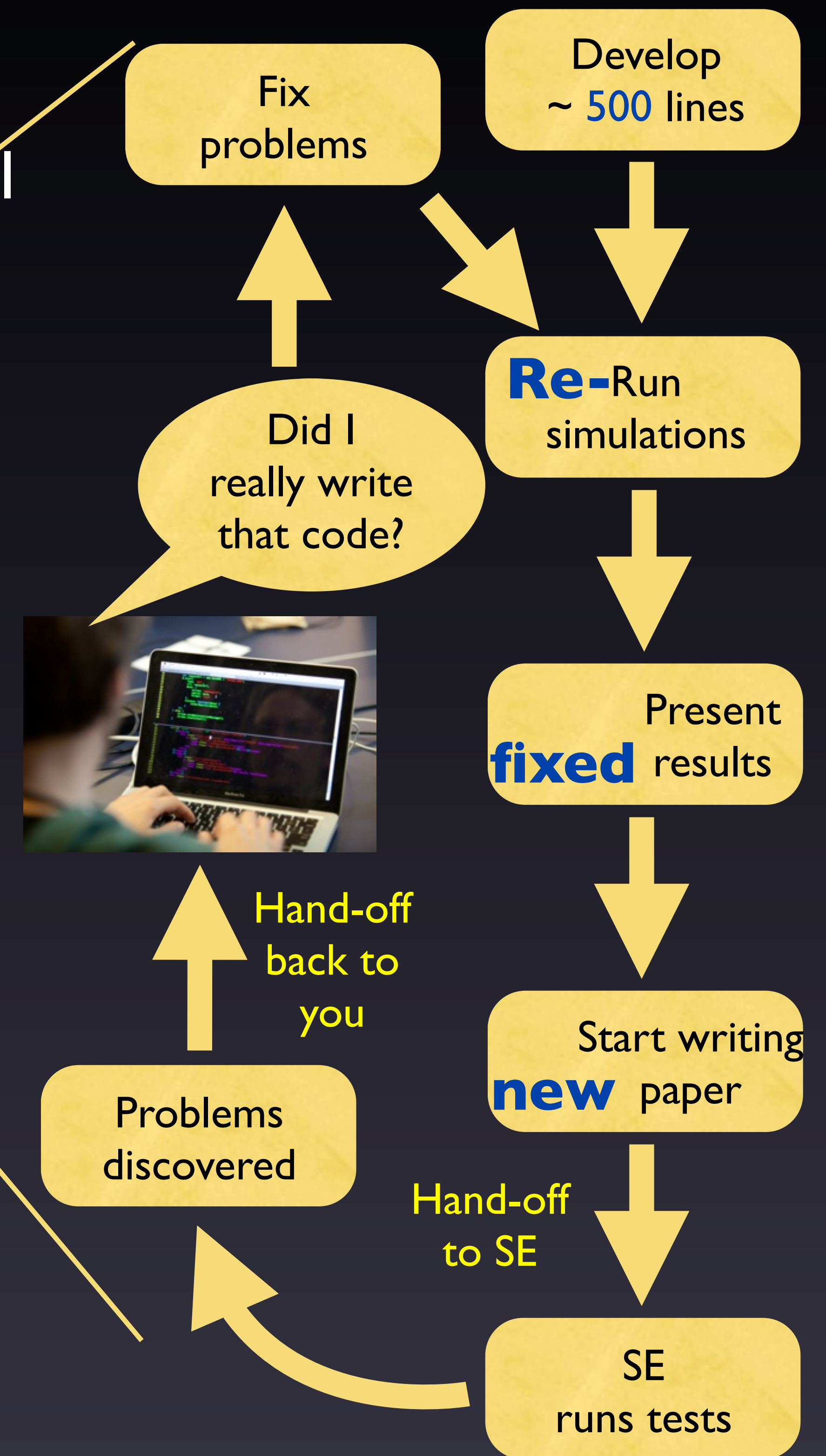
Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

Left-hand flow chart: time frame of cycle ~ days

Isn't Testing the Responsibility of Software Engineers?



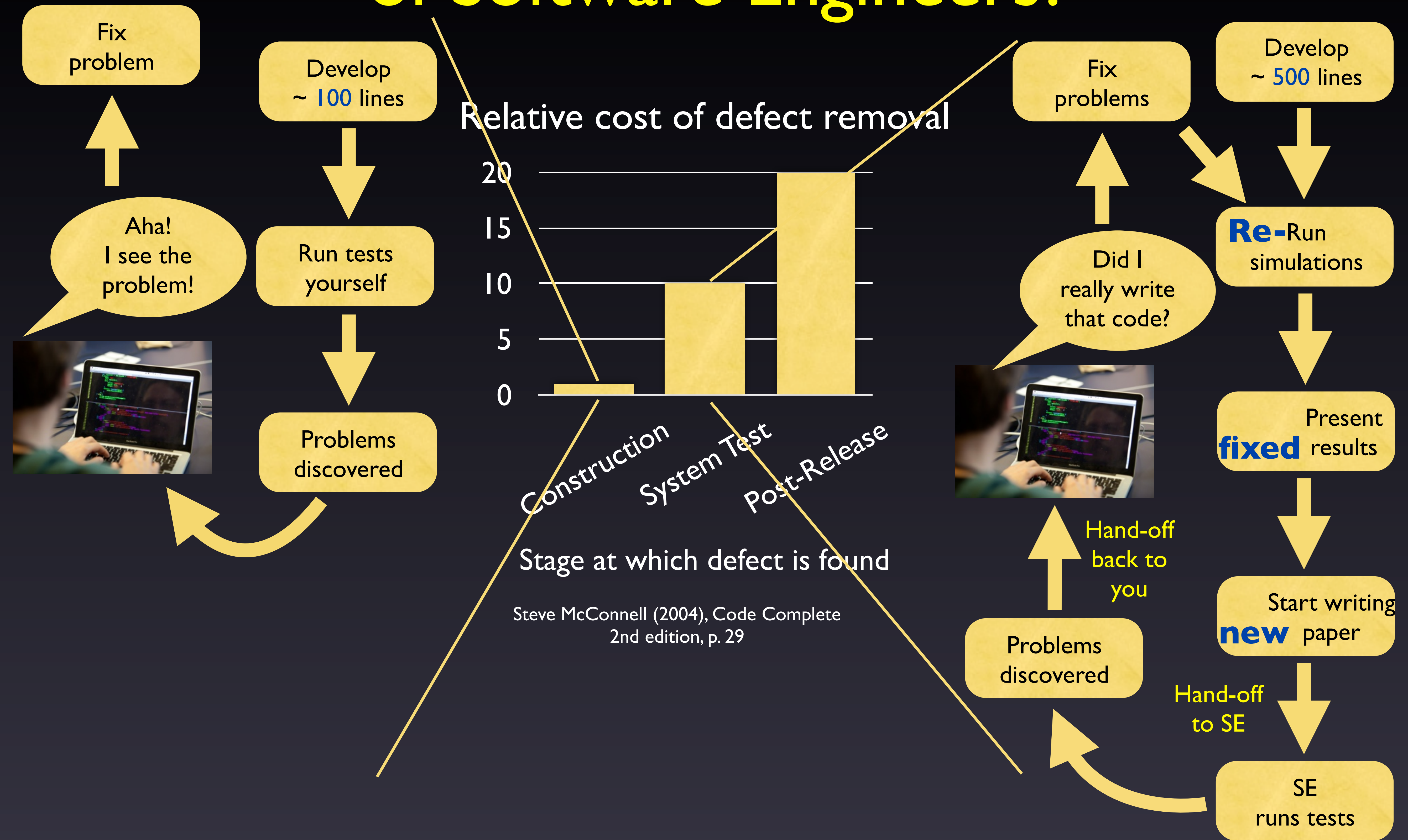
Steve McConnell (2004), Code Complete
2nd edition, p. 29



Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

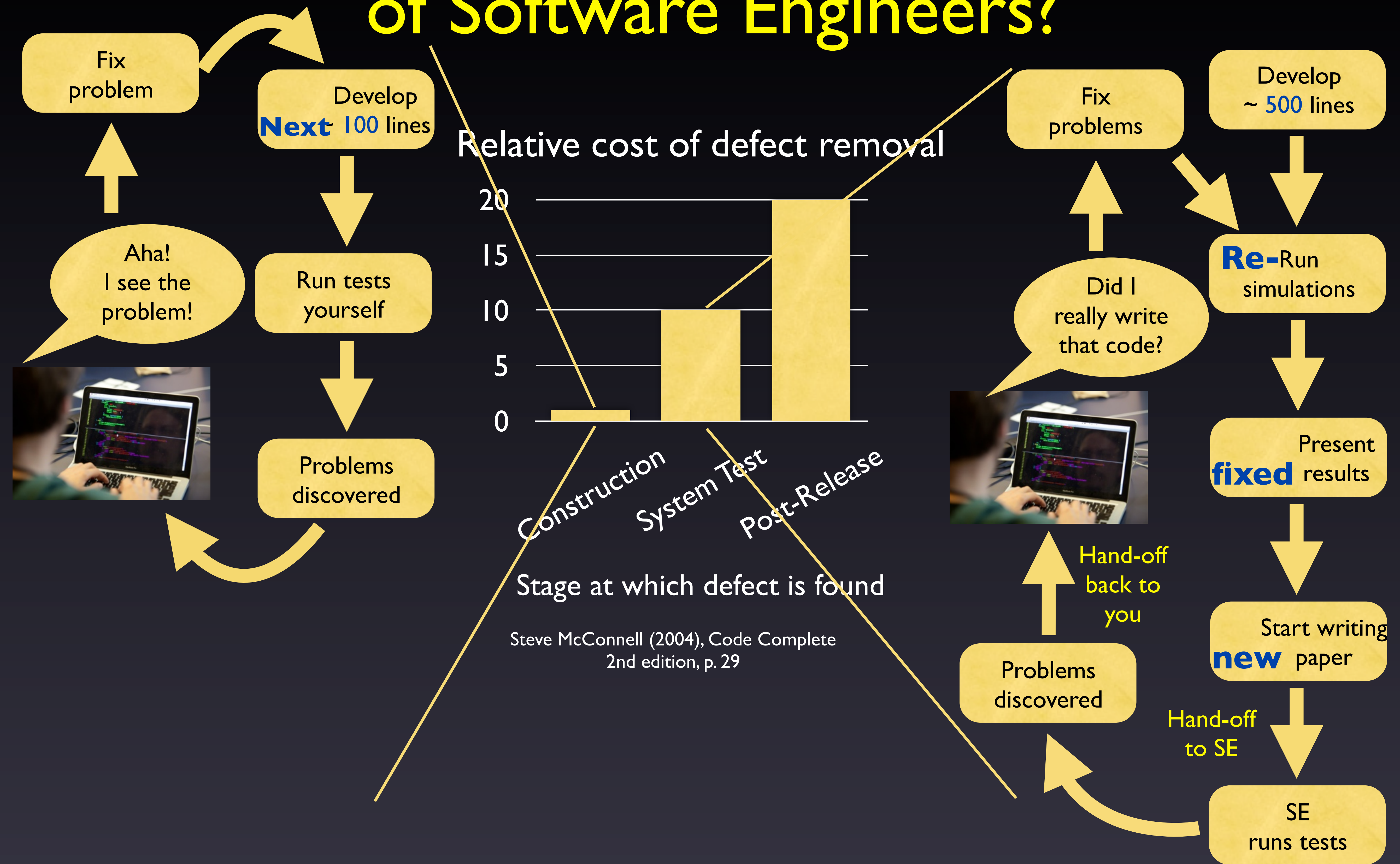
Left-hand flow chart: time frame of cycle ~ days



Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

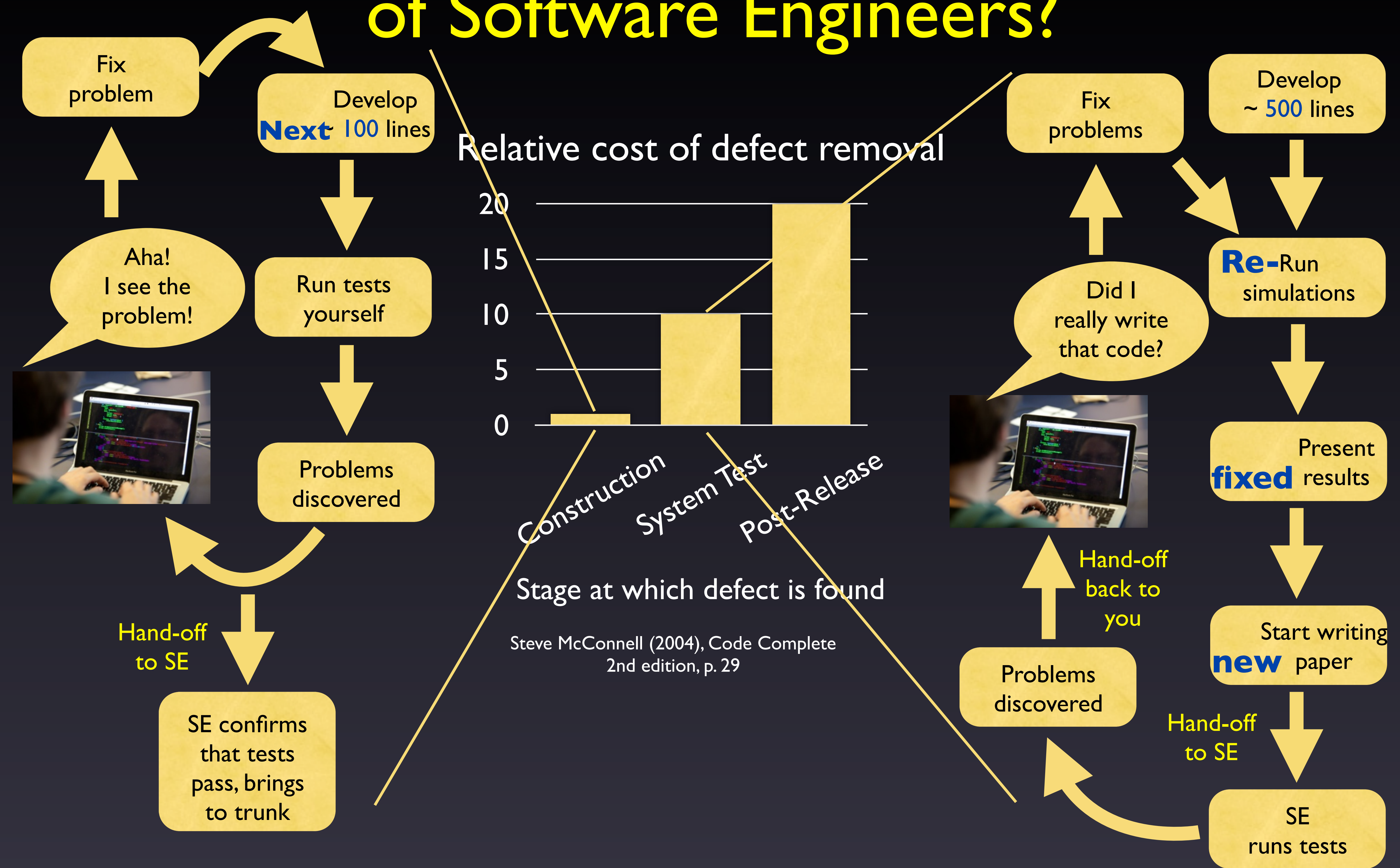
Left-hand flow chart: time frame of cycle ~ days



Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

Left-hand flow chart: time frame of cycle ~ days



Take-away from the graph: if you catch your own problems early, it will speed up bringing your code to the trunk

Right-hand flow chart: time frame ~ months (or years). Sources of inefficiency:
- forgetting what you have done
- Lots of changes -> hard to find the source of problems
- May need to redo experiments, etc.

Left-hand flow chart: time frame of cycle ~ days

What CESM's Test System Can Do for You

- Single tests that you run frequently while developing
- Pre-built test lists that you run periodically, which test various functionality across many configurations
- Automated comparisons with baselines for non-answer-changing modifications

Frequent single tests: ~ daily

Big test lists: maybe ~ weekly

What CESM's Test System Can NOT Do for You

- Is your code correct? This is the role of:
  - Manual tests – some of which should later be added to the automated test suite so nobody breaks YOUR code
  - Unit tests – framework now in place in CESM
- Power diminished when you have answer-changing modifications
  - Try to break your development into multiple stages, separating answer-changing from bit-for-bit changes

Answer-changing: OR: bring in your changes controlled by a flag in the model, and ensure that you truly just change answers when that flag is turned on

Outline

- Intro & motivation
- Basics of using the automated test system
- Comparing against baselines
- Running a whole test suite
- Summary
- Appendix: References for later use

How to Run a Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
```

How to Run a Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
```

How to Run a Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
```

OPTIONAL: Unique ID for a given testname.
If not given, defaults to YYMMDD-HHMMSS

Test type
(ERS: exact restart)

How to Run a Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
```

OPTIONAL: Unique ID for a given testname.
If not given, defaults to YYMMDD-HHMMSS

Test type
(ERS: exact restart)

OPTIONAL: Extra test options
(_D: turn on debug flags)
(separate multiple options with _)

How to Run a Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
```

OPTIONAL: Unique ID for a given testname. If not given, defaults to YYMMDD-HHMMSS

Test type (ERS: exact restart)

OPTIONAL: Extra test options (_D: turn on debug flags) (separate multiple options with _)

Resolution

Compset

Machine

Compiler

Examples here are for yellowstone, but this works the same on any supported machine.

How to Run a Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
```

OPTIONAL: Unique ID for a given testname. If not given, defaults to YYMMDD-HHMMSS

Test type (ERS: exact restart)

OPTIONAL: Extra test options (_D: turn on debug flags) (separate multiple options with _)

Resolution

Compset

Machine

Compiler

Examples here are for yellowstone, but this works the same on any supported machine.

Case name = testname.testid

```
cd ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01

./ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01.test_build
./ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01.submit
```

How to Run a Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
```

OPTIONAL: Unique ID for a given testname.
If not given, defaults to YYMMDD-HHMMSS

Test type
(ERS: exact restart)

Resolution

Compset

Machine

Compiler

OPTIONAL: Extra test options
(_D: turn on debug flags)
(separate multiple options with _)

Examples here are for yellowstone, but this works the same on any supported machine.

Case name = testname.testid

```
cd ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01

./ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01.test_build
./ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01.submit
```

Note use of test_build rather than standard build script.
This is important, because the test_build script sometimes does additional work.

Common Test Types

Functionality to Test	Test Type
Runs to completion	SMS (smoke test)
Restarts bit-for-bit	ERS (exact restart test)
Hybrid / branch / restarts bit-for-bit	ERI (ERS on steroids; can be hard to debug)
Results independent of processor count	PEM (PE counts MPI bit-for-bit)
Threading	PET (with & without threading bit-for-bit)
Compilation with debug flags (check array bounds, floating point trapping, etc.)	Add _D option
Longer run (default is typically 5 days)	Add _L option (_Lm3 = 3 months, _Ly5 = 5 years, etc.)

For a complete list, run the following from $CCSMROOT/scripts:

```
ccsm_utils/Testlistxml/manage_xml_entries -list tests
```


Yellow = main test types you'll want to run

Checking Test Results

```
cd $CCSMROOT/scripts/
ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01

cat TestStatus
```


Image credits:
http://openclipart.org/detail/30217/tango-face-smile-by-warszawianka
http://openclipart.org/detail/30223/tango-face-sad-by-warszawianka

Checking Test Results

```
cd $CCSMROOT/scripts/
ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01

cat TestStatus


PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01.memleak
```

Image credits:
http://openclipart.org/detail/30217/tango-face-smile-by-warszawianka
http://openclipart.org/detail/30223/tango-face-sad-by-warszawianka

Checking Test Results

```
cd $CCSMROOT/scripts/
ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01

cat TestStatus
```

```
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01.memleak
```

Or you might see:

```
FAIL  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01
```

See TestStatus.out file for more details of the failure.

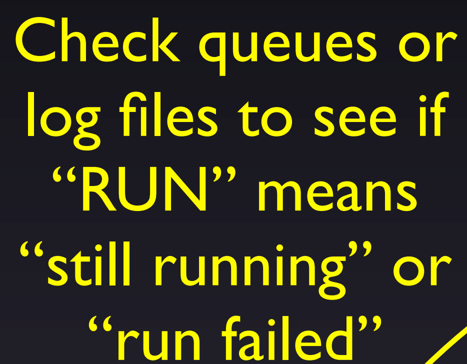
(See Appendix for ideas on where to look first for different failure types.)

Image credits:
http://openclipart.org/detail/30217/tango-face-smile-by-warszawianka
http://openclipart.org/detail/30223/tango-face-sad-by-warszawianka

Common Result Codes

Result Code	Meaning
Good results	
PASS	Test passed
Bad results	
TFAIL	Test setup error
SFAIL	Generation of test failed in scripts
CFAIL	Build error
RUN	Run timed out or exited abnormally
FAIL	Test failed (either due to run failure or, e.g., non-exact restarts for an ERS test)
Test not yet complete	
GEN	Test has been generated
BUILD	Build succeeded, not yet submitted
PEND	Test submitted, waiting in queue
RUN	Test is currently running

Common Result Codes



Result Code	Meaning
Good results	
PASS	Test passed
Bad results	
TFAIL	Test setup error
SFAIL	Generation of test failed in scripts
CFAIL	Build error
RUN	Run timed out or exited abnormally
FAIL	Test failed (either due to run failure or, e.g., non-exact restarts for an ERS test)
Test not yet complete	
GEN	Test has been generated
BUILD	Build succeeded, not yet submitted
PEND	Test submitted, waiting in queue
RUN	Test is currently running

Sample TestStatus.out: ERS Failures

TestStatus = RUN
Initial run failed

```
doing a 11 ndays initial test
pass = 0
ERROR in /var/spool/torque/mom_priv/jobs/16682.goldbach.cgd.ucar.edu.SC:
coupler log indicates that inital model run failed
```

Sample TestStatus.out: ERS Failures

TestStatus = RUN
Initial run failed

```
doing a 11 ndays initial test
pass = 0
ERROR in /var/spool/torque/mom_priv/jobs/16682.goldbach.cgd.ucar.edu.SC:
coupler log indicates that inital model run failed
```

TestStatus = FAIL
Run succeeded, but restart wasn't bit-for-bit

```
doing a 11 ndays initial test
pass = 1
doing a 5 ndays restart test
Initial Test log is /scratch/cluster/sacks/ERS_D.f10_f10.ICLM45BGC.goldbach_intel.t01/
run/cpl.log.140312-125941
Restart Test log is /scratch/cluster/sacks/ERS_D.f10_f10.ICLM45BGC.goldbach_intel.t01/
run/cpl.log.140312-130327
Initial Test hist is /scratch/cluster/sacks/ERS_D.f10_f10.ICLM45BGC.goldbach_intel.t01/
run/ERS_D.f10_f10.ICLM45BGC.goldbach_intel.t01.cpl.hi.0001-01-12-00000.nc.base
Restart Test hist is /scratch/cluster/sacks/ERS_D.f10_f10.ICLM45BGC.goldbach_intel.t01/
run/ERS_D.f10_f10.ICLM45BGC.goldbach_intel.t01.cpl.hi.0001-01-12-00000.nc
Comparing initial log file with second log file
Difference found beginning at 10107 1800 :
< comm_diag xxx sorr   1 2.1942676188259493750E+14 recv lnd Sl_avsdr
> comm_diag xxx sorr   1 2.1971003083939603125E+14 recv lnd Sl_avsdr
< comm_diag xxx sorr   2 2.1806167094445437500E+14 recv lnd Sl_anidr
...
FAIL
```

Making Arbitrary Configuration Changes to a Test

- What we have shown so far only allows you to test out-of-the-box compsets
- There is also a capability to change any xml variable or namelist option
  - Done via a "testmods" directory, containing user_nl files and/or a file of xmlchange commands
- Example:

Note extra component in the test name

```
create_test -testname
ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.clm-ciso
```

- For details, see slides in Appendix

Intention is: everything should be specified via the test name or the testmods directory – you should NOT need to make any manual changes in your test's case directory. This is to (1) facilitate rerunning a given test, and (2) comparing against baselines

Outline

- Intro & motivation
- Basics of using the automated test system
- Comparing against baselines
- Running a whole test suite
- Summary
- Appendix: References for later use

At this point, you may be thinking, “okay, great, I can do most of this myself, with create_newcase. Why do I need this test system?” And to some extent you would be right. But I hope that, with the next two sections, you’ll see the advantage of using automated tests.

Purpose of Baseline Comparisons

Make sure answers haven't changed; this can mean:

- No answers change, e.g., if you are doing an answer-preserving code refactoring
- Some answers change, e.g., if you change CLM-crop code, and want to make sure that answers are still bit-for-bit for runs without crop

Purpose of Baseline Comparisons

- Because we don't have many testable specifications of how CESM should work, baseline comparisons are the strongest tool available to make sure you haven't broken anything

- To take full advantage of this tool, try to separate your changes into:
  - Bit-for-bit modifications that can be tested against baselines
    - e.g., renaming variables and moving code around, either before or after your science changes
  - Answer-changing modifications
    - Try to make these as small as possible, so that they can be more easily reviewed for correctness

Yes, this is more work, but it's worth it to prevent the introduction of bugs

Baseline Comparisons
Step 1: Determine if you need to generate baselines

- Decide what to use as a baseline
  - Generally a trunk version, or a previous, well-tested version of your branch
- Determine if you need to generate baselines
  - If comparing against a trunk version, baselines may exist (e.g., on yellowstone, see `$CESMDATAROOT/ccsm_baselines` for CESM & CLM baselines)
  - Otherwise, you'll need to generate your own baselines

Baseline Comparisons
Step 2: Generate baselines

(Skip this step if baselines already exist
for the desired baseline code version)

- Check out the baseline code version
- Run create_test from the baseline code with the -generate option:

```
mkdir /glade/p/work/$USER/cesm_baselines

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-generate clm4_5_59
```

Baseline Comparisons
Step 2: Generate baselines

Confirming that baselines have been successfully generated

```
cd $CCSMROOT/scripts/ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01

cat TestStatus

PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01.memleak
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01.generate.clm4_5_59
```


Note G added to case name (‘generate’)

Baseline Comparisons
Step 2: Generate baselines

Confirming that baselines have been successfully generated

```
cd $CCSMROOT/scripts/ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01

cat TestStatus

PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01.memleak
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.G.t01.generate.clm4_5_59


ls  /glade/p/work/$USER/cesm_baselines/clm4_5_59/
ERS_D.f10_f10.ICLM45BGC.yellowstone_intel

CaseDocs   cpl.log.140312-153410  user_nl_clm   user_nl_rtm
cpl.hi.nc  cpl.log.140312-154007  user_nl_cpl
cpl.log    TestStatus.out         user_nl_datm
```

Comparisons will be done using this coupler history file, which contains fields passed between components. Note that individual component history files are NOT compared, but you can add those comparisons using the component_gen_comp tool (see Appendix).

Note G added to case name (‘generate’)

Baseline Comparisons
Step 3: Compare against baselines

- Run create_test from your modified code with the -compare option (and -generate too, if desired):

Baseline Comparisons
Step 3: Compare against baselines

- Run create_test from your modified code with the -compare option (and -generate too, if desired):

```
./create_test -testid t02
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59
-generate mynew_clm4_5_59
```

It doesn't hurt to generate new baselines: it's easier to remove them than it is to generate baselines after the fact. Just be sure to give your new baselines a meaningful name, which differs from any existing baselines for this testname.

Interpreting Baseline Comparisons

Comparisons Pass

```
cd $CCSMROOT/scripts/ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.C.t02

cat TestStatus

PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.memleak
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.generate.mynew_clm4_5_59
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.compare_hist.clm4_5_59
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.memcomp.clm4_5_59
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.tputcomp.clm4_5_59
PASS ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.nlcomp
```

- compare_hist: Main comparison: FAIL means coupler history files differ
- memcomp: FAIL means memory use increased significantly
- tputcomp: FAIL means run time increased significantly
  - Lots of false positives: You can generally ignore this
- nlcomp: FAIL means component namelists differ

Note GC added to case name (generate & compare)

Interpreting Baseline Comparisons

Comparisons Fail

```
cd $CCSMROOT/scripts/ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02

cat TestStatus

PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.memleak
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.generate.mynew_clm4_5_59
FAIL  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.compare_hist.clm4_5_59
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.memcomp.clm4_5_59
FAIL  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.tputcomp.clm4_5_59
COMMENT   tput_decr = 9.791 tput_percent_decr = 17.3
FAIL ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.nlcomp
```

Excerpt from TestStatus.out:

```
Comparing hist file with baseline hist file
...

SUMMARY of cprnc:
 A total number of    170 fields were compared
          of which     38 had non-zero differences
               and      0 had differences in fill patterns
 A total number of      0 fields could not be analyzed
 A total number of      0 fields on file 1 were not found on file2.
  diff_test: the two files seem to be DIFFERENT

FAIL
hist file comparison is FAIL
```

For full differences, view cprnc.out in your case directory
(search for RMS in that file to see fields that differ)

Interpreting Baseline Comparisons

Missing baselines

```
cd $CCSMROOT/scripts/ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02

cat TestStatus

PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.memleak
PASS  ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.generate.mynew_clm4_5_59
BFAIL ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t02.compare_hist.clm4_5_59
```

Outline

- Intro & motivation
- Basics of using the automated test system
- Comparing against baselines
- Running a whole test suite
- Summary
- Appendix: References for later use

Recap: What Do We Want to Test?

"I didn't break any other functionality"

- Make sure other model configurations still work
  - Example: Making sure CLM still works when you turn on prognostic crops
- Make sure code works with other compilers
- If you expect a set of changes to maintain identical answers, make sure that's true
  - Terminology: "Bit-for-bit"

All of this is a lot to remember. And that's the motivation for the automated test suite: Then you don't have to remember to test all of these different things. In fact, you don't even have to know what threading is. The automated test suite will test these things for you and tell you if there is a problem.

Running a Test Suite

- Allows running many tests with a single command
- Create your own test suite
  - Simply a text file listing all the tests you want to run
  - See Appendix for an example
- Run a pre-built test suite
  - Allows you to test many configurations, including ones you have never heard of!

Pre-Built Test Lists

```
ccsm_utils/Testlistxml/manage_xml_entries -list categories
```

Commonly-used categories:

- aux_clm: Used when making a CLM trunk tag
- aux_clm_short: Small subset of aux_clm, for more frequent testing
- aux_glc: Used when making a GLC trunk tag
- prealpha: Used when making a CESM alpha tag
- prebeta: Used when making a CESM beta tag

Note that this does NOT cover the standard CAM testing (which uses its own test system), but you can do plenty of CAM testing using this standard CESM test system

You should talk to a CSEG member for tips on an appropriate test list to run for your development

Viewing a Pre-Built Test List

```
ccsm_utils/Testlistxml/manage_xml_entries -query -outputlist
    -category aux_clm -mach yellowstone -compiler intel
```

Viewing a Pre-Built Test List

```
ccsm_utils/Testlistxml/manage_xml_entries -query -outputlist
    -category aux_clm -mach yellowstone -compiler intel
```

mach & compiler are optional
Exclude these options to see all tests in this category

Viewing a Pre-Built Test List

```
ccsm_utils/Testlistxml/manage_xml_entries -query -outputlist
  -category aux_clm -mach yellowstone -compiler intel
```

mach & compiler are optional
Exclude these options to see all tests in this category

- SMS.f45_f45.I.yellowstone_intel.clm-ptsRLA
- SMS.f45_f45.I.yellowstone_intel.clm-ptsROA
- ERS_E.f19_g16.I1850.yellowstone_intel
- PET_P180x2_D.f19_g16.I1850CLM45.yellowstone_intel
- CME_Ly4.f10_f10.I1850CLM45BGC.yellowstone_intel.clm-monthly
- CME_N2.f10_f10.I1850CLM45BGC.yellowstone_intel.clm-default
- ERS.f19_g16.I1850CLM45BGC.yellowstone_intel.clm-default
- ERS_D_E.f19_g16.I1850CLM45BGC.yellowstone_intel.rtm-rtmOnIceOff
- ERS_E.f19_g16.I1850CRUCLM45CN.yellowstone_intel.clm-default
- SMS.1x1_mexicocityMEX.I1PTCLM45.yellowstone_intel.clm-default
- ERS_Lm3.1x1_vancouverCAN.I1PTCLM45.yellowstone_intel.clm-default
- SMS_D.1x1_mexicocityMEX.I1PTCLM50.yellowstone_intel.clm-default
- ERS_Lm3.1x1_vancouverCAN.I1PTCLM50.yellowstone_intel.clm-default
- SMS_Ly3.1x1_tropicAtl.I20TRCLM45BGC.yellowstone_intel.clm-tropicAtl_subsetLate
- SMS_Ly5.1x1_tropicAtl.I20TRCLM45BGC.yellowstone_intel.clm-tropicAtl_subsetMid
- SMS_Ly8.1x1_tropicAtl.I20TRCLM45BGC.yellowstone_intel.clm-tropicAtl_subsetEarly
- ERI_D.f10_f10.I20TRCN.yellowstone_intel
- ERS_Ly5.f10_f10.I20TRCRUCLM45BGC.yellowstone_intel.clm-monthly_noinitial
- ERI_D.T31_g37.ICLM45.yellowstone_intel.clm-SNICARFRC
- SMS_D_Mmpi-serial.f45_f45.ICLM45.yellowstone_intel.clm-ptsRLA
- SMS_Mmpi-serial.f45_f45.ICLM45.yellowstone_intel.clm-ptsRLA
- ERI.f09_g16.ICLM45BGC.yellowstone_intel
- ERI_D.f09_g16.ICLM45BGC.yellowstone_intel
- ERI.f10_f10.ICLM45BGC.yellowstone_intel
- ERI_D.f10_f10.ICLM45BGC.yellowstone_intel
- ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.clm-rootlit
- ERI.f19_g16.ICLM45BGC.yellowstone_intel
- ERI_D.f19_g16.ICLM45BGC.yellowstone_intel
- ERI_D.ne30_g16.ICLM45BGC.yellowstone_intel.clm-vrtlay
- ERI_D.ne30_g16.ICLM45BGC.yellowstone_intel
- ERS_Ly5.f10_f10.ICLM45BGCCROP.yellowstone_intel.clm-irrigOn_reduceOutput
- PET_P15x2_Ly3.f10_f10.ICLM45BGCCROP.yellowstone_intel.clm-irrigOn_reduceOutput
- SMS_Ly1.f19_g16.ICLM45BGCCROP.yellowstone_intel
- PET_P15x2_Lm25.f10_f10.ICLM45BGCDVCROP.yellowstone_intel.clm-reduceOutput
- ERS_D.f19_g16.ICLM45GLCMEC.yellowstone_intel.clm-glcMEC_changeFlags
- ERS_D.f09_g16.ICLM45VIC.yellowstone_intel.clm-vrtlay
- ERS_D.f10_f10.ICLM45VIC.yellowstone_intel.clm-vrtlay
- SMS.f19_g16.ICLM45VIC.yellowstone_intel.clm-default
- CME.f10_f10.ICN.yellowstone_intel
- ERS_Ld3_D_P64x16.ne30_g16.ICN.yellowstone_intel
- PET_D_P4x30.ne30_g16.ICN.yellowstone_intel
- ERS_Ld211_D_P112x1.f10_f10.ICNCROP.yellowstone_intel.clm-crop
- ERS_Ld211_P192x1.f19_g16.ICNDVCROP.yellowstone_intel.clm-crop
- NCK.f10_f10.ICRUCLM45.yellowstone_intel
- ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel
- ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.clm-default
- ERI.f10_f10.ICRUCLM50BGC.yellowstone_intel
- ERI_D.f10_f10.ICRUCLM50BGC.yellowstone_intel
- ERI.f19_g16.ICRUCLM50BGC.yellowstone_intel
- ERI_D.f19_g16.ICRUCLM50BGC.yellowstone_intel
- ERS_Lm3.f19_g16.IGRCP60CN.yellowstone_intel
- SMS_Ld5.f19_g16.IRCP45CLM45BGC.yellowstone_intel.clm-decStart

Running a Pre-Built Test List

```
./create_test -testid t01
-xml_category aux_clm
-xml_mach yellowstone -xml_compiler intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59
-generate mynew_clm4_5_59
```


This is really cool. If you have started to drift off, now is the time to start paying attention again

Just like for single tests, you may need to run a separate test suite with ‘generate’ first to generate baselines

Need to run a separate command for each compiler: this may change soon

Running a Pre-Built Test List

```
./create_test -testid t01
-xml_category aux_clm
-xml_mach yellowstone -xml_compiler intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59
-generate mynew_clm4_5_59
```

- This one command creates all the tests on the previous slide, then builds and submits them for you

This is really cool. If you have started to drift off, now is the time to start paying attention again

Just like for single tests, you may need to run a separate test suite with ‘generate’ first to generate baselines

Need to run a separate command for each compiler: this may change soon

Running a Pre-Built Test List

```
./create_test -testid t01
-xml_category aux_clm
-xml_mach yellowstone -xml_compiler intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59
-generate mynew_clm4_5_59
```

- This one command creates all the tests on the previous slide, then builds and submits them for you!!!

This is really cool. If you have started to drift off, now is the time to start paying attention again

Just like for single tests, you may need to run a separate test suite with ‘generate’ first to generate baselines

Need to run a separate command for each compiler: this may change soon

Running a Pre-Built Test List

```
./create_test -testid t01
-xml_category aux_clm
-xml_mach yellowstone -xml_compiler intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59
-generate mynew_clm4_5_59
```

- This one command creates all the tests on the previous slide, then builds and submits them for you!!!
  - This command can take a while to complete; see Appendix for workflow hints
- Need to run a separate command for each compiler
  - e.g., for aux_clm, run a second command for pgi on yellowstone

This is really cool. If you have started to drift off, now is the time to start paying attention again

Just like for single tests, you may need to run a separate test suite with ‘generate’ first to generate baselines

Need to run a separate command for each compiler: this may change soon

Checking Results from a Test Suite

create_test creates a script named cs.status.$testid.$machine
Run this script to check test results for all tests in the test suite

```
./cs.status.t01.yellowstone
```

Checking Results from a Test Suite

create_test creates a script named cs.status.$testid.$machine
Run this script to check test results for all tests in the test suite

```
./cs.status.t01.yellowstone
```

Small excerpt:

```
PASS  CME.f10_f10.ICN.yellowstone_intel.C.t01
PASS  CME.f10_f10.ICN.yellowstone_intel.C.t01.generate.mynew_clm4_5_59
PASS  CME.f10_f10.ICN.yellowstone_intel.C.t01.compare_hist.clm4_5_59
PASS CME.f10_f10.ICN.yellowstone_intel.C.t01.nlcomp
PASS  CME_Ly4.f10_f10.I1850CLM45BGC.yellowstone_intel.clm-monthly.C.t01
PASS  CME_Ly4.f10_f10.I1850CLM45BGC.yellowstone_intel.clm-monthly.C.t01.generate.mynew_clm4_5_59
PASS  CME_Ly4.f10_f10.I1850CLM45BGC.yellowstone_intel.clm-monthly.C.t01.compare_hist.clm4_5_59
PASS CME_Ly4.f10_f10.I1850CLM45BGC.yellowstone_intel.clm-monthly.C.t01.nlcomp
...
FAIL  ERI_D.ne30_g16.ICLM45BGC.yellowstone_intel.clm-vrtlay.C.t01
PASS  ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.C.t01
PASS  ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.C.t01.memleak
PASS  ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.C.t01.generate.mynew_clm4_5_59
PASS  ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.C.t01.compare_hist.clm4_5_59
PASS  ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.C.t01.memcomp.clm4_5_59
PASS  ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.C.t01.tputcomp.clm4_5_59
PASS ERI_N2.f19_g16.ICRUCLM45BGCCROP.yellowstone_intel.C.t01.nlcomp
...
```

Checking Results from a Test Suite

- Rerun the cs.status script as often as you want, to view results as they come in
  - At first you'll see a lot of GEN results
- Investigating failures is the same as for single tests
  - Go into relevant test directory, look at TestStatus.out, etc.
- Note that there may be some expected failures
  - See if the failing test passed in the baseline code
  - Or talk to the relevant CSEG member

Note that there may be some expected failures. e.g., for aux_clm, see https://wiki.ucar.edu/display/ccsm/CLM+Testing; for aux_glc see the CISM ChangeLog; for prealpha and prebeta, see https://csegweb.cgd.ucar.edu/testdb/cgi-bin/login.cgi

So Running a Huge Test Suite Is as Easy as 1-2-3

So Running a Huge Test Suite Is as Easy as 1-2-3

1) ./create_test -testid t01 -xml_category aux_clm -xml_mach yellowstone -xml_compiler intel -baselineroot /glade/p/work/$USER/cesm_baselines -compare clm4_5_59 -generate mynew_clm4_5_59

So Running a Huge Test Suite Is as Easy as 1-2-3

1)
```
./create_test -testid t01 -xml_category aux_clm
-xml_mach yellowstone -xml_compiler intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59 -generate mynew_clm4_5_59
```

2)
```
./cs.status.t01.yellowstone
```

So Running a Huge Test Suite Is as Easy as 1-2-3

1)
```
./create_test -testid t01 -xml_category aux_clm
-xml_mach yellowstone -xml_compiler intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59 -generate mynew_clm4_5_59
```

2)
```
./cs.status.t01.yellowstone
```

3) Celebrate all of your passing tests!

Outline

- Intro & motivation
- Basics of using the automated test system
- Comparing against baselines
- Running a whole test suite
- Summary
- Appendix: References for later use

Summary

- Automated testing lets you catch bugs sooner, speeding development
- CESM's automated test suite facilitates:
  - Quick tests that you can run frequently
  - Full test suites of lots of configurations that you can run periodically
- You now have the following testing tools at your disposal:
  - Single tests
    - Basic "smoke" tests
    - Tests of requirements like exact restart
  - Test suites that you create yourself (see Appendix)
  - Pre-built test suites
  - All of which allow comparisons to baselines, to make sure answers only change when you expect them to change

Outline

- Intro & motivation
- Basics of using the automated test system
- Comparing against baselines
- Running a whole test suite
- Summary
- Appendix: References for later use

Contents of Appendix

- Where to go for more information
- What CESM versions does this cover?
- Full example: single test
- Where to look if your test fails
- Full example: test suite
- Recommendation for test suites: use ‘screen’
- Rerunning failed tests in a test suite
- Running a test suite on a different machine
- Details of using a testmods directory
- Defining your own test list
- Comparing component history files with component_gen_comp

Where to Go for More Information

- Slides & recording from this talk
  - http://www2.cgd.ucar.edu/sections/cseg/tutorials
- Chapter 7 of the CESM User's Guide
  - http://www.cesm.ucar.edu/models/cesm1.2/cesm/doc/usersguide/book1.html
  - Note that some of this is outdated – we no longer have query_tests (replaced by manage_xml_entries)
- CLM's guide to testing
  - https://wiki.ucar.edu/display/ccsm/CLM+Testing
- Interactive help for tools discussed here
  - `create_test -help`
  - `ccsm_utils/Testlistxml/manage_xml_entries -help`
  - `ccsm_utils/Tools/component_gen_comp -help`

What CESM Versions Does This Cover?

- In general, I refer to the latest development code
- Much of this is the same in the CESM1.2 release
- Single tests: Main functionality has been the same for a while
- Test suites: Functionality has been in place for a while, but command-line syntax changed significantly in CESM1.2
  - And the command to query a test list has changed even more recently than that
- Note that examples are for yellowstone, but you can use these tools on any machine

--- NOTES FOR THE READER ---

In the CESM1.2 release code, use $CESMROOT/scripts/query_tests in place of $CESMROOT/scripts/ccsm_utils/Testlistxml/manage_xml_entries

Note that the machine needs to have the cprnc tool installed, and the test scripts need to know where to find it (this is the case for CESM-supported machines)

Full Example: Single Test

```
cd $CCSMROOT/scripts

./create_test -testid t01
-testname ERS_D.f10_f10.ICLM45BGC.yellowstone_intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59
-generate mynew_clm4_5_59

cd ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t01
```

No cesm_setup needed (create_test does that for you)

```
./ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t01.test_build
./ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.GC.t01.submit
```

Wait for test to finish

```
cat TestStatus
```

If the test failed:

```
less TestStatus.out

cd /glade/scratch/$USER/ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.t01/run

less lnd.log.*
less cesm.log.*
```

Where to Look If Your Test Fails

Failure Code	Where to Look First
TFAIL	Output from create_test
SFAIL	Output from create_test
CFAIL	Output from test_build script (will generally point you to a build log file)
RUN	(1) Batch log files in case directory: determine if it simply ran out of wall-clock time (2) TestStatus.out file in case directory (3) Log files and core files in run directory Note: Some tests (e.g., ERI) create multiple run directories, with .ref1, .ref2 extensions; you may need to check all of them, e.g., check: ERI.f10_f10.ICLM45BGC.yellowstone_intel.t01.ref1/run
FAIL	(1) TestStatus.out file in case directory: this will help you see the cause of failure – e.g., run didn't complete vs. test requirements (such as exact restart) weren't met. Look for FAIL in this file, and any messages above the FAIL line. (2) If run didn't complete, check log files and core files in run directory (3) If requirements of test weren't met, TestStatus.out will generally refer to differences in coupler log files and/or coupler history files. History file differences can be seen in the cprnc.out file in the run directory.

Full Example: Test Suite

```
./create_test -testid t01.intel -xml_category aux_clm
-xml_mach yellowstone -xml_compiler intel
-baselineroot /glade/p/work/$USER/cesm_baselines
-compare clm4_5_59 -generate mynew_clm4_5_59  | tee t01.intel.out
```

"Pipe" (send) the output into the "tee" command.
tee is a unix command that copies all of the terminal output into the given file (t01.intel.out).
This allows easier viewing of the output later
– e.g., you can search this file for tests that had SFAIL or CFAIL results.

```
./cs.status.t01.intel.yellowstone | grep -v -e PASS -e tputcomp -e COMMENT
```

"Pipe" (send) the output into a "grep" command, which excludes all lines containing "PASS", "tputcomp", or "COMMENT".
These lines can generally be ignored. Thus, what you'll see are lines requiring your attention, such as FAIL results.

Recommendation for Test Suites: Use the 'screen' command

- Motivation: Building and running tests is time-consuming, requires the developer to keep a long-running terminal session open.
- What is screen?: Unix command that "virtualizes" a terminal session. Sessions can be created, then detached and reattached from different machines.
- screen -S 'yellowstonetest': Creates a session with the specified name.
- screen -ls: Lists the currently open screen sessions.
- screen -d -r 'yellowstonetest': Attaches to screen session, detaching it if already attached.
- Testing workflow:
  - At work: For each machine, start a screen session either locally or on remote machine.
  - Check out code, start tests.
  - Later, at home: Reattach to screen sessions, check on test status.

Rerunning Failed Tests in a Test Suite

- If lots of tests failed, generally easiest to rerun the test suite from scratch
  - Give it a new testid
- If just a few tests failed, due to system problems or minor bugs
  - Official recommendation is to re-create these failed tests from scratch, as individual tests, or by creating your own test suite
    - This is the safest thing to do
  - But often it will work to go into the case directories of the failed tests, and rerun the test_build and submit scripts

Running a Test Suite on a Different Machine

Example: You want to run all of the aux_clm tests that are normally run on yellowstone with the pgi compiler, but you want to run them on the machine 'edison' with the intel compiler

```
./create_test -testid t01.intel -xml_category aux_clm
-xml_mach yellowstone -xml_compiler pgi
-mach edison -compiler intel
```

xml_mach and xml_compiler say, "find the test list set up for this machine and compiler".
By default, the machine and compiler used for the tests is the same.
But you can override that by specifying the -mach and/or -compiler options.

Details of Using a Testmods Directory

- Any namelist changes or xml variable changes can be made using a testmods directory
- This directory contains either or both:
  - user_nl files for any component(s)
    - e.g., user_nl_clm, user_nl_cam
    - Just like the user_nl files in a case, these can have any namelist changes
  - A file called xmlchange_cmnds containing commands used to change xml variables
    - This can contain any number of lines with commands to run, such as: `./xmlchange RUN_STARTDATE=2001-12-30`
- By default, this directory should go in scripts/ccsm_utils/Testlistxml/testmods_dirs
  - See directories in there for examples
  - The default location can be changed using the -user_testmods_dir option to create_test

Details of Using a Testmods Directory

Use your testmods directory by specifying an extra component in your testname:

```
create_test -testname
ERS_D.f10_f10.ICLM45BGC.yellowstone_intel.clm-ciso
```

This gives the path to the testmods directory.
The path is relative to scripts/ccsm_utils/Testlistxml/testmods_dirs,
unless the -user_testmods_dir option is given to create_test.
Note that subdirectories are separated by '-'
– i.e., use a dash in place of '/' when separating directory components of the path.

Defining Your Own Test List

- You can easily run your own list of tests
- To do this, simply create a text file, with one test name per line
  - i.e., each line would be the 'testname' argument to create_test
- You can then run your whole test list similarly to how you run pre-built test lists.
  - But don't use any of the -xml_* options to create_test (xml_category, xml_mach, xml_compiler)
  - Instead, use the -input_list option to create_test
    - e.g., `create_test -input_list my_test_list ...`
    - (where my_test_list is the text file you created)
- Note that a given test list should only use a single machine & compiler

Comparing Component History Files with component_gen_comp

- Recall that, when doing baseline comparisons, only coupler history files are compared
- Sometimes you want to compare component history files (e.g., CLM and/or CAM history files), to make sure diagnostic fields haven't changed
- This can be done with scripts/ccsm_utils/Tools/component_gen_comp
- Run this after your test suite has completed
- Need to specify the following options:
  - -baselineroot, -generate, -compare: Same as the options to create_test
  - -testid: testid of the test suite that you just ran, from which you want to generate or compare component history files
  - -model: name of component to generate / compare
    - Currently just set up for clm (give it the model name clm2), cism and cpl
    - Could easily be extended to other components
  - -runloc: path to directory containing test run directories
- First you will need to run it with the -generate option to generate baselines, then you can run it with the -compare option to compare against those baselines
- Note that this will only be effective if your tests generate component history files. This can be done by running longer tests (e.g., > 1 month), or by using a testmods directory that specifies more frequent history output.
- Note that BFAIL1 results from -compare can be ignored: these generally indicate that there simply weren't any component history files for this test
- Run 'ccsm_utils/Tools/component_gen_comp -help' for more details