How Long Does Flight Software Testing Take?

Dan Houston and Man Lieu
The Aerospace Corporation

10 December 2010

How long will testing take?

- The need for estimating test duration
  - *Testing is a discovery process, very difficult to estimate*
  - *Defect discovery ("reliability growth") curves are often used to assess readiness ("Are we there yet?")*
  - *The difficulty of estimation is exacerbated by …*
    - software complexity (increases time to analyze defects)
    - system constraints (increases likelihood of defects and difficulty of finding them, *e.g.*, timing problems in real-time software)
    - reliability requirements (removes ability to postpone quality)
  - *The question of software availability becomes more important with the difficulty of moving release dates due to …*
    - resources required for product launch
    - customers relying on a service switchover

Software readiness can be a highly important and very challenging issue

Some approaches to the question

- Linear estimates
  - *Allocate a number of hours per test case and multiply by the number of test cases*
  - *Estimate the total time to run all test cases without stopping and multiply by an "estimated average" of the number of test runs*
- Expert judgment plus Monte Carlo sampling
  - *Ask for estimates of best case, worst case, and most likely case estimates of test duration; produce triangular distributions, weight them and randomly sample them many times to produce an aggregate distribution*
- Estimation tools
  - *Use a software development project estimation tool to estimate the project duration, then allocate a fraction to the testing phase*

Can you identify the flaw(s) in each of these methods?

More on the need for a better representation

- Problems with these approaches to the question
  - *Linear estimates ignore the structure of the process, a test-and-fix cycle*
  - *Expert judgment plus Monte Carlo sampling breaks the first rule of survey research: asking people for information they don't have*
  - *Using tools developed for project-level estimation means "reading into" phase-level activities*
- Other reasons for simulating a test process
  - *Understand and explain the dynamics of the resource-constrained test-and-fix process*
  - *Learn what input values are necessary to meet a desired schedule*
  - *Assess the value of alternative scenarios*
  - *Provide a clear, objective basis for recommendations, even if they could have been made without modeling*
  - *Figure out what data is important to collect for forecasting*

We need a method that represents the process well

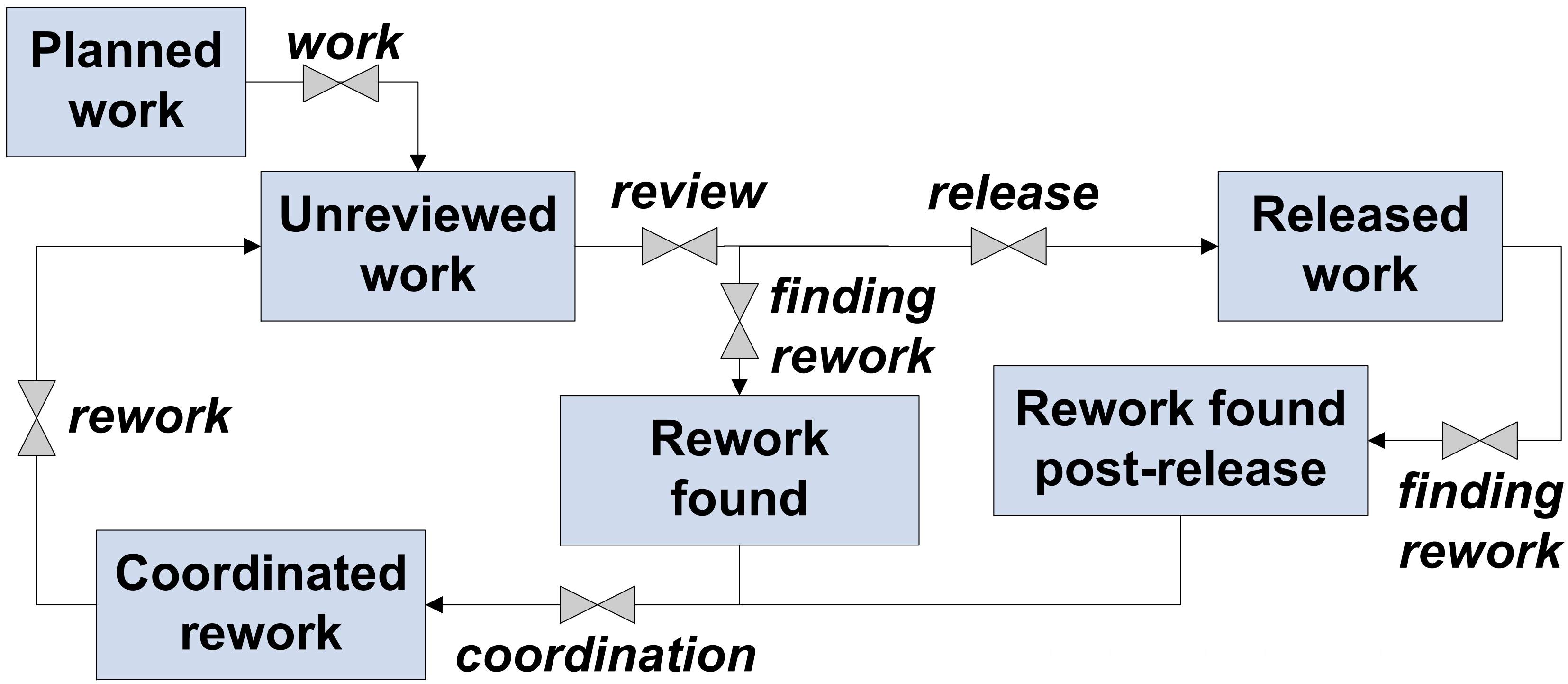
Applying process simulation

- Many software process models include a testing phase in which rework is a single-pass activity, *e.g.*
  - *Abdel-Hamid and Madnick (1991)*
  - *Tvedt (1996)*
  - *Zhang, et al. (2008)*
- Others represent development and rework as cyclic
  - *Cooper (1993)*
  - *Ford and Sterman (1998)*
- Flows include software, defects, and test cases
- A multi-phase system dynamics model
  - *Based on Ford and Sterman (1998)*
  - *Designate the flow as test cases*
- Real-time software testing requires use of special test facilities
  - *These test facilities constrain the testing process, creating a "bottleneck"*
  - *Simplified the problem: didn't need to represent human resources*

How rework is modeled depends on the process and the question to be answered!

Multi-phase system dynamics model

Focuses on initial work (upper left) – in this case, testing – and rework loops – in this case, retesting (bottom left).

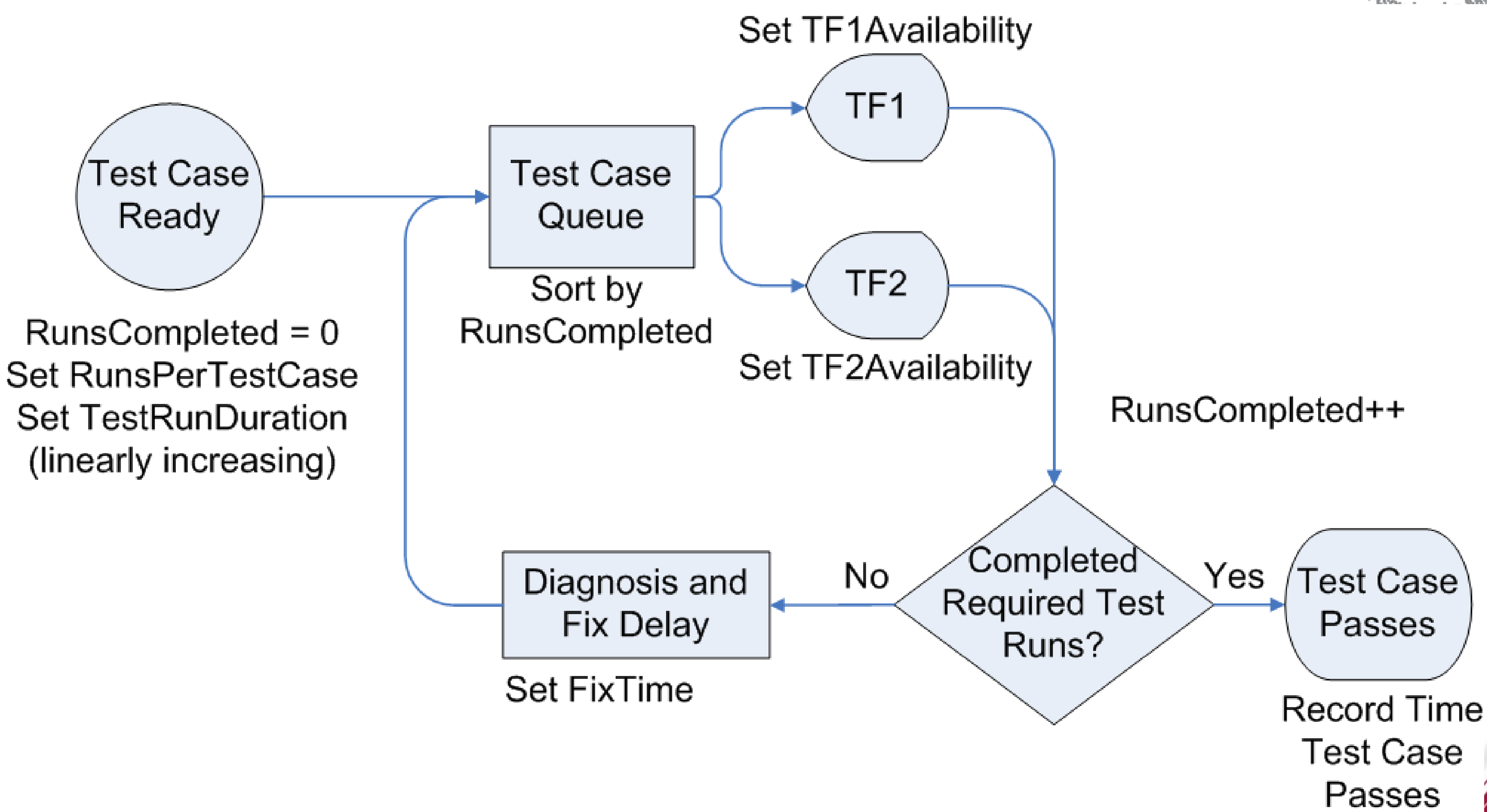


Model is arrayed across 3 phases: test cases review, test-and-fix, and final test

Looking beyond the system dynamics model

- Resource constraint for real-time testing
  - *Can be represented but not as clearly as in a discrete event model*
- Level of measurement problem
  - *Ford's model assumes an atomic level*
  - *Cannot represent entire test cases circulating*
- Factors modeled in a discrete event model
  - *Percentage of time each test facility is available*
  - *Delay in re-running a test case due to fixing defects*
  - *Average number of test sessions required for each test to reach maturity*
    - This factor operationalizes test interruptions for many reasons: test script problems, software defects, lab configuration problems, *etc*.
  - *Average duration of lab occupancy for running a test case*
    - Includes fixed time for testing a case (preparation and setup for running a case, results capture and storage, cleanup, etc.) plus time required to run a case

Test-and-Fix (TaF) Duration Model



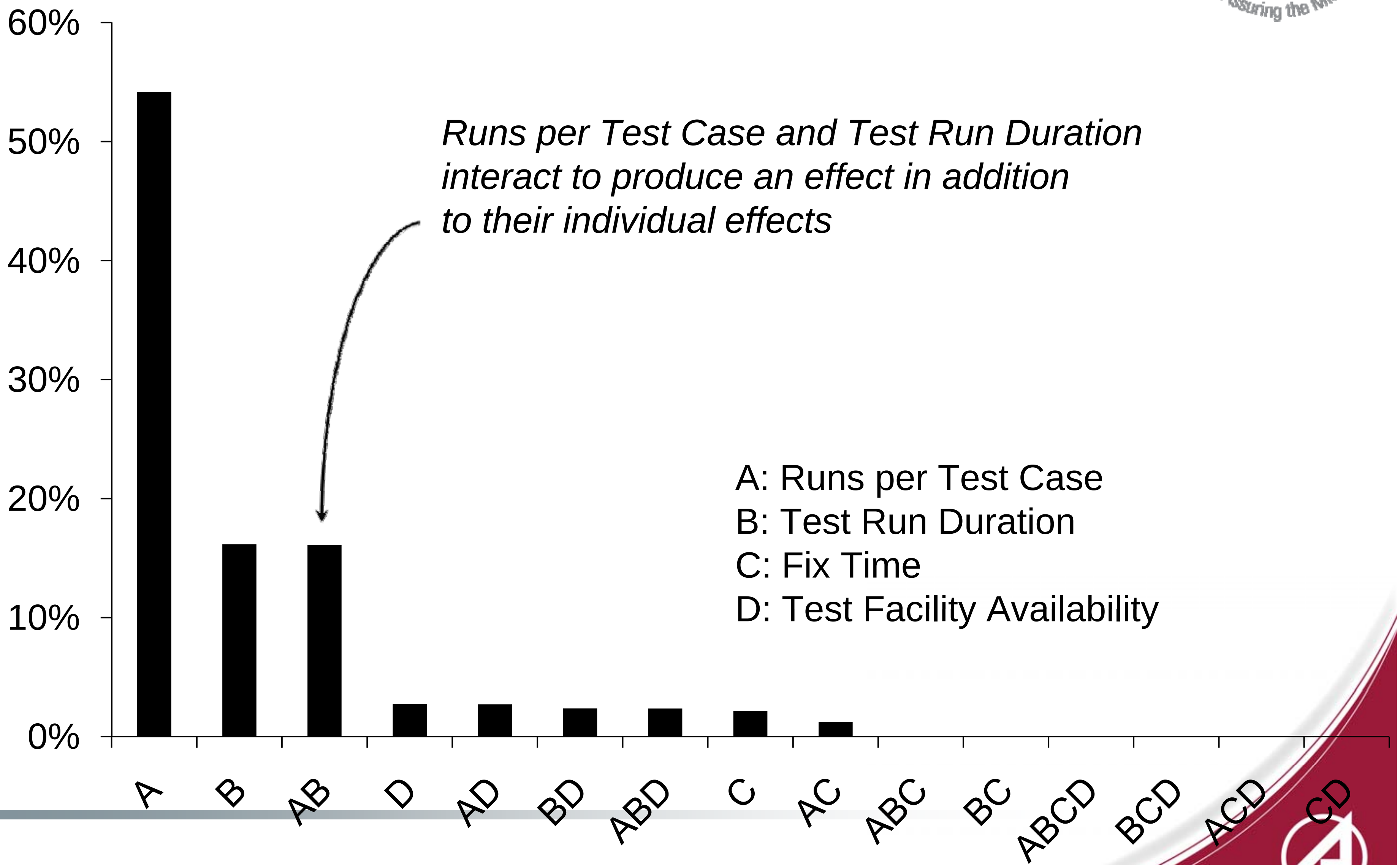
Discovering significant factors

- Used a full factorial experiment
  - *Use constant inputs representing expected operational values*
  - *All combinations of four factors at two levels each (2^4): 16 simulation runs*
  - *Response variable is duration of TaF process*

Factor	**Low Value**	**High Value**
Test Facility Availability	60 hrs/ week	100 hrs/week
Runs per Test Case	2	8
Test Run Duration	2 hrs	5 hrs
Fix Time	24 hrs	96 hrs

- Analysis of variance
  - *Calculate percentage contribution to variation in duration from sums of squares*

Significant factors

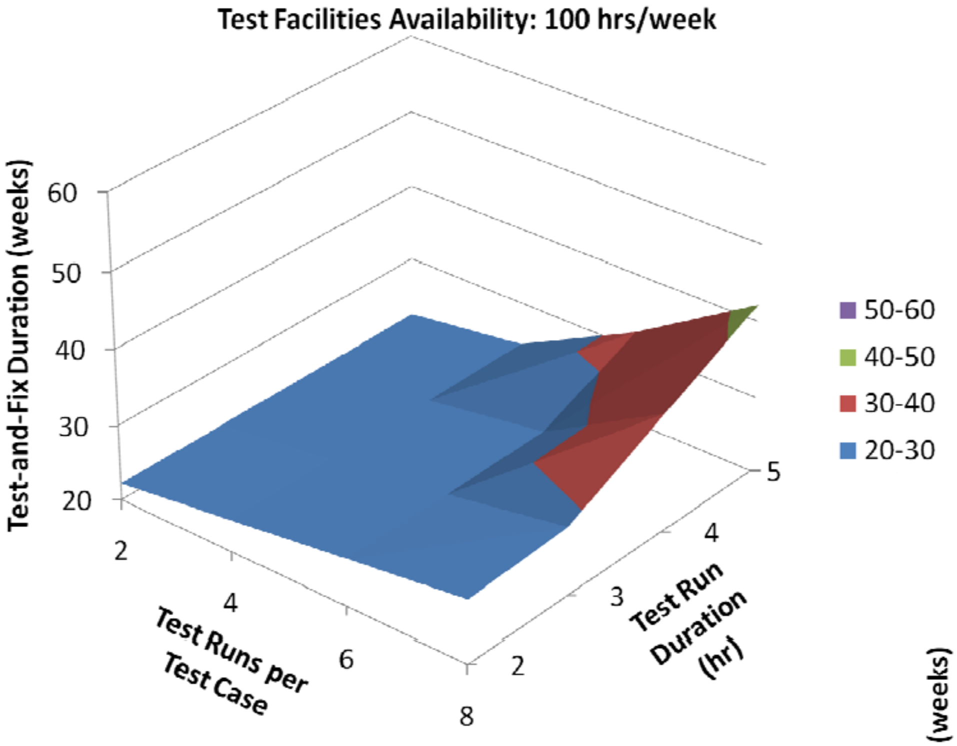


Discovering behavior

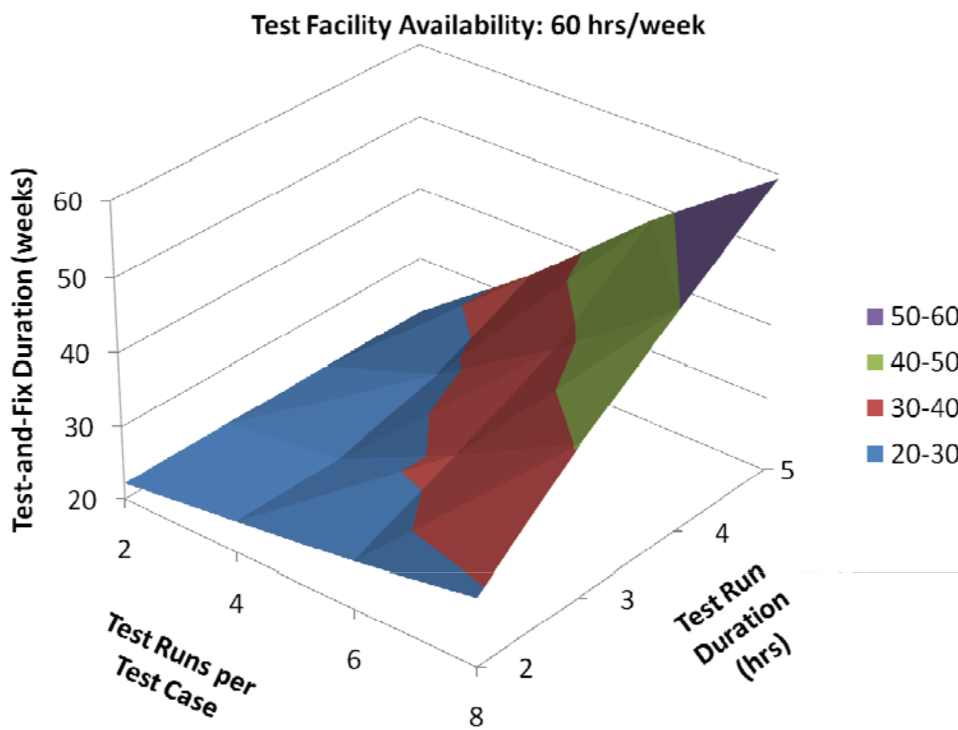
- Used an additional full factorial experiment to produce response surfaces
  - *Focus on Runs per Test Case and Test Run Duration*
  - *Use one Fix Time value and two Test Facility Availability values*

Factor	**Values**
Test Facility Availability	60 and 100 hrs/week
Runs per Test Case	2, 4, 6, 8
Test Run Duration	2, 3, 4, 5 hrs
Fix Time	7 days

Behavior: the TF threshold



Factor interaction above the TF full utilization threshold



TF availability moves the threshold

Modeling a likely scenario and alternatives

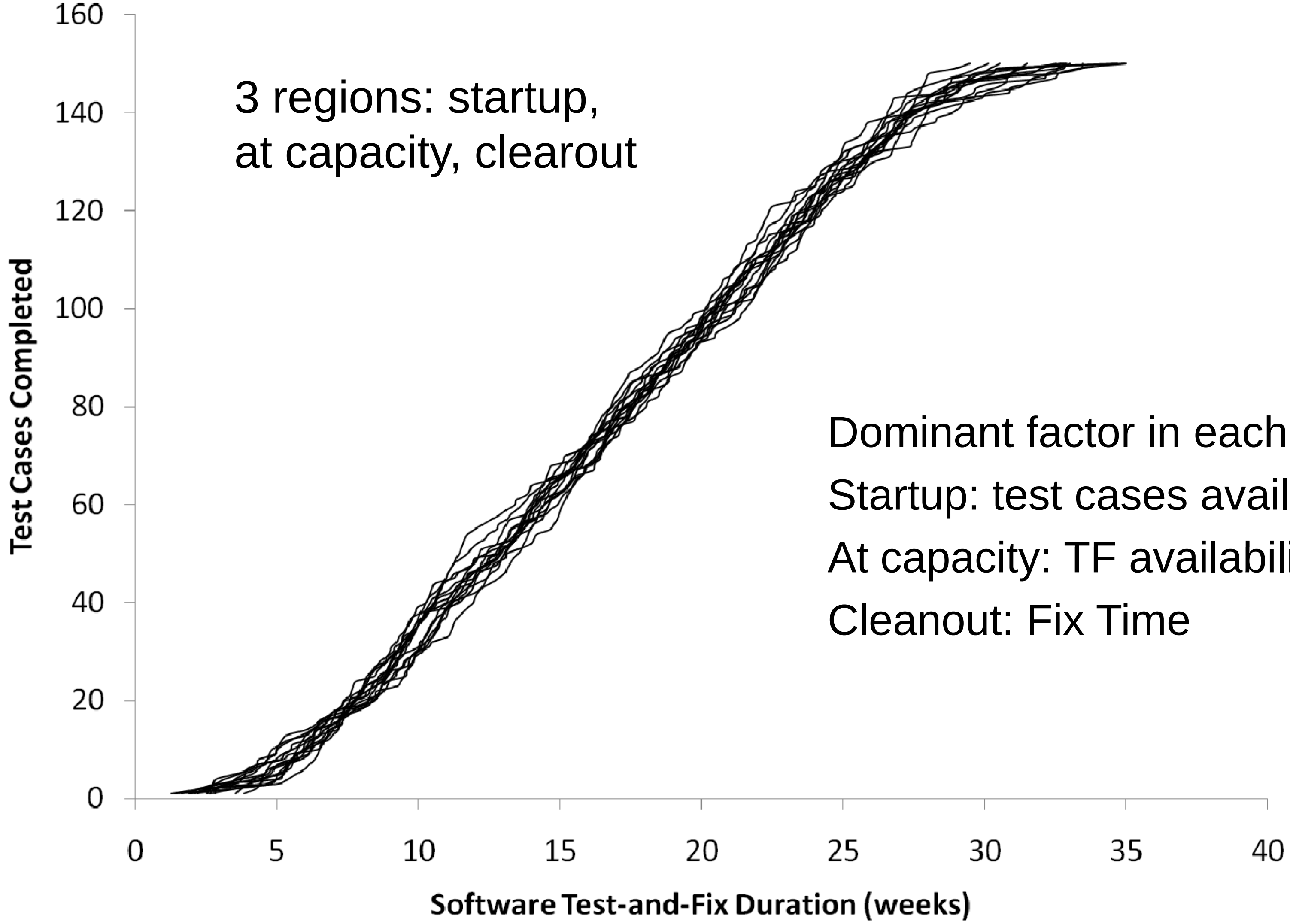
- Used likely inputs to estimate the duration of the test-and-fix cycle

Factor	Values	Sample
Test Facility Availability	Both test facilities at 40 hrs/week each	Constant for all simulation runs
Runs per Test Case	(2, .1), (3, .1), (4, .3) (5, .2), (6, .1) (7, .05), (8,.05)	Randomly for each test case in each simulation run
Test Run Duration	Triangular(2, 3.5, 5) hrs	Randomly for each test case in each simulation run
Fix Time	(7, .125), (8, .125), (9, .125), (10, .125), (11, .125), (12, .125), (13, .125), (14, .125) days	Randomly for each test cycle of each test case in each simulation run

- Alternative scenarios
  - *Additional test facility availability or an additional test facility*
  - *More optimistic Test Run Duration and/or Fix Time*

Test case completion times

3 regions: startup, at capacity, clearout



Dominant factor in each region

Startup: test cases availability

At capacity: TF availability

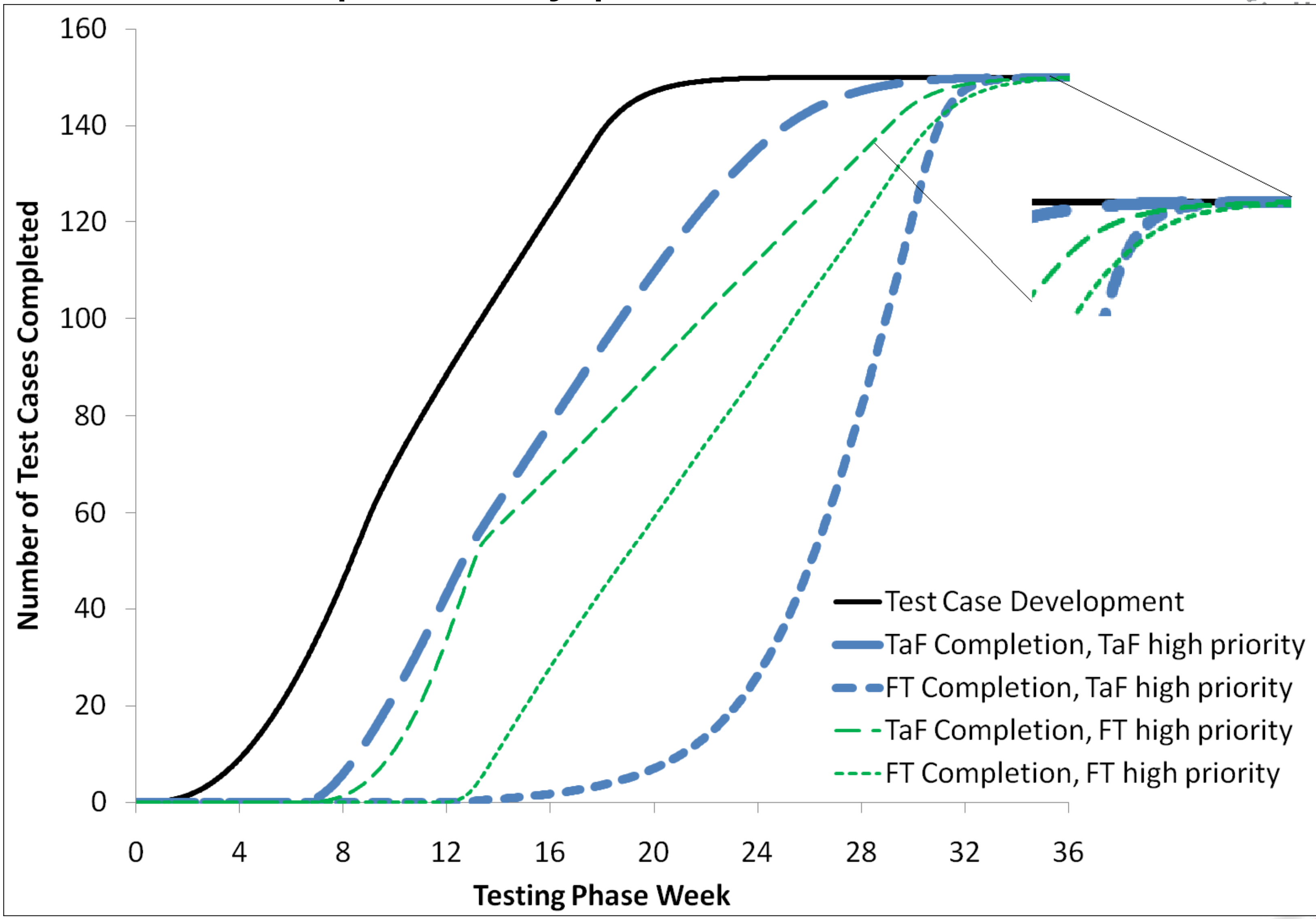
Cleanout: Fix Time

Back to the multi-phase model

- Take the results from the single-phase model and use it to calibrate the multi-phase model
  - *Assume that test cases development will complete as planned*
  - *Set the test-and-fix cycle to last ~30 weeks*
  - *Assume that half of the test cases will need 2 test sessions in final test*

- Run three cases of this scenario
  - *Give test-and-fix priority over final testing for test facility use*
  - *Let test-and-fix and final test contend equally for test facility use*
  - *Give final testing priority over test-and-fix for test facility use*

Test case completion by phase



Study Recommendations

- Conduct well-performed quality-inducing activities—for example, peer reviews, unit testing, and defect causal analysis—prior to employing the test facilities. The modeling provided case-based, quantitative support for this quality cost principle.
- Reduce the number of test sessions. If a test case fails, continue running it as far as possible in order to find further anomalies. Reduce diagnostic time through good anomaly documentation. This trade-off in favor of reducing the number of test runs comes at the expense of the less important factor, test run duration.
- Reduce the fixed time of test runs (setup, recordkeeping, saving files). Automation can significantly reduce time in a test facility, as well as facilitating good documentation of anomalies.
- Reduce the learning time of testers through training and regular communication.
- The efficiency of a test team can be undermined by individuals optimizing their own tasks.
- Complete TaF before FT. This suggests that showing progress early may carry a cost in the form of a longer overall duration.
- As the end of TaF approaches, focus on reducing the time to provide fixes. As fewer test cases are left in the test-and-fix cycle, fix delays can dominate the cycle duration.

Conclusions

- Identified Runs per Test Case as the dominant factor in test-and-fix phase duration rather than Test Facility Availability
  - *It is a quality issue rather than a facilities issue*
- Test Facility Availability magnifies the effects of other factors
- Collect Runs per Test Case and Test Run Duration data for improving forecast of test completion
- Simulation accounts for delays and factor interactions when estimating test duration
- Competition for test facility time between phases could delay project completion
- Input values necessary to meet a desired schedule may be infeasible
- The benefit of adding test facilities depends on the timing of availability but is usually low

References

- Dörner, D.: The Logic of Failure: Recognizing and Avoiding Error in Complex Situations. Perseus Publishing, Cambridge, Massachusetts (1996)
- Kellner, M., Madachy, R., Raffo, D.: Software Process Simulation Modeling: Why? What? How? Journal of Systems and Software 46:2/3, 91-105 (1991)
- Abdel-Hamid, T.K., Madnick, S.E.: Software Project Dynamics. Prentice Hall, Englewoord Cliffs, New Jersey (1991)
- Tvedt, J.D.: An Extensible Model for Evaluating the Impact of Process Improvements of Software Development Time. PhD dissertation. Arizona State University (1996)
- Zhang, H.,Jeffrey, R., Zhu, L.: Hybrid Modeling of Test-and-Fix Processes in Incremental Development. In Wang, Q., Pfahl, D., Raffo, D. (eds.) Masking Globally Distributed Software Development a Success Story, International Conference on Software Process, ICSP 2008, Leipzig, Germany. Springer-Verlag, Berlin Heidelberg.
- Cooper, K.G., Mullen, T.W.: Swords and plowshares: The rework cycle of defense and commercial software development projects. American Programmer 6:5, 41-51 (May 1993)
- Ford, D.N., and Sterman, J.D.: Dynamic Modeling of Product Development Processes. System Dynamics Review 14:1, 31-68 (Spring 1998)