

Junior Johnson vs. the Dwarves

Brian Beckman

<2017-04-05 Wed>

Contents

| | |
|---|----------|
| 1 | 3 |
| 2 Deliver the Hooch! | 4 |
| 2.1 Deadline! Drive Like Junior Johnson | 4 |
| 2.2 Gotta be lucky AND good | 5 |
| 3 Choice 2: Dig a Tunnel | 5 |
| 3.1 You only have to be good | 5 |
| 4 Summary & Disclaimer | 6 |
| 4.1 WIP: Not fully thought out | 6 |
| 4.2 As Always: "Unless You Know Better" | 6 |
| 5 A Story | 6 |
| 5.1 I'm digging along | 6 |
| 6 A Story | 6 |
| 6.1 Tuesday Night, slam-coding basics from a paper | 6 |
| 7 A Story | 7 |
| 7.1 Wednesday -> Friday, Three Thousand More Lines | 7 |
| 7.2 Friday Night, Integration Time, Something is Wrong | 7 |
| 8 A Story | 7 |
| 8.1 But C / C++ / Python / MATLAB / etc. say | 7 |
| 8.2 Unit Testing would have caught this Tuesday | 7 |
| 8.3 Putting off testing to Friday requires us to debug / search | 7 |
| 9 The Tradeoff | 7 |
| 9.1 If you don't need predictable schedule and high assurance | 7 |
| 9.2 Otherwise, you need predictable schedule or high assurance | 8 |

| | |
|--|-----------|
| 10 The Tradeoff | 8 |
| 10.1 TDD trades $O(N)$ dev time for $O(N \log N)$ debugging time | 8 |
| 10.2 Debugging is SEARCH | 8 |
| 10.3 TDD is LINEAR | 8 |
| 11 Unit Tests vs. PBT (Property-Based Testing) | 10 |
| 11.1 Unit Tests | 10 |
| 11.2 Property-Based Testing (PBT) | 10 |
| 12 Examples: | 10 |
| 12.1 Kalman Filter in C / C++ / Python | 10 |
| 12.2 An Interview Question | 10 |
| 12.3 Time Warp Operating System | 10 |
| 13 Abstract: | 10 |
| 13.1 Test-Driven Development (TDD) can deliver higher-quality results with documen- tation | 10 |
| 13.2 and more predictable schedules than can slam-coding and debugging, but it can take longer. | 10 |
| 13.3 It's like digging a tunnel to deliver the moonshine versus driving at 100 mph through | 10 |
| 13.4 the woods at night with the headlights off. We present tradeoff analysis and examples | 10 |
| 13.5 from embedded aviation code in C / C++, where certification authorities often require | 10 |
| 13.6 traceable documentation and high assurance (machine-checked coding standards; proofs; | 10 |
| 13.7 exhaustive or statistical unit testing; more). We also present examples in Clojure, which | 10 |
| 13.8 facilitates lightweight formal specs and generative testing for JVM code. | 10 |
| 13.9 Normal unit testing is based on point-like examples invented by programmers (one point | 10 |
| 13.10in input domain mapped to one point in the output range). It's limited by the ability of | 10 |
| 13.11programmers to invent examples that exercise everything pertinent to a spec. | 10 |
| 13.12Property-based testing generates broader tests, statistically sampling the input do- mains | 10 |
| 13.13and checking assertions about properties of outputs. It's limited by the ability of . . . | 10 |
| 13.14programmers to understand broader properties and to imagine non-obvious tests. . . | 10 |

1



2 Deliver the Hooch!

2.1 Deadline! Drive Like Junior Johnson

2.1.1 100 mph in the woods at night with the headlights off



2.2 Gotta be lucky AND good

2.2.1 hit a tree? you have a problem

2.2.2 no telling how long till you're going again

2.2.3 you might not make it

3 Choice 2: Dig a Tunnel



3.1 You only have to be good

3.1.1 takes longer, but no backtracking and search / debugging

1. time is more predictable

3.1.2 lots of time for certification, documentation, proofs (“high assurance”)

4 Summary & Disclaimer

4.1 WIP: Not fully thought out

4.1.1 I reserve the right to change my mind

4.1.2 TDD “feels” better because it sidesteps debugging

1. Makes every programming language feel like a REPL
2. Unpredictable debugging pitfalls are avoided more frequently
3. Easily defensible for exploratory designs
4. Defensible when large test corpora are required for certification
5. Less defensible under time pressure

4.1.3 Irony: defensible at the extremes of development

1. When you’re not sure what you’re doing (“science”)
2. When you’re absolutely sure what you’re doing (“engineering with high QA”)

4.1.4 I have VERY recent experiences to talk about

4.2 As Always: “Unless You Know Better”

5 A Story

5.1 I’m digging along

5.1.1 on schedule to deliver certifiable nav filter in three weeks

5.1.2 phone call! “We’re flying Wednesday! Deliver MONDAY!”

6 A Story

6.1 Tuesday Night, slam-coding basics from a paper

$$B = \begin{cases} \frac{1}{2} - \frac{\theta^2}{4!} + \frac{\theta^4}{6!}, & |\theta| \lesssim 10^{-7} \\ (1 - \cos \theta) / \theta^2 & \text{otherwise} \end{cases}$$

if (std::abs(th) < SMALL_ANGLE) { B = (0.5 - th * th * 1.0 / 24.0

- th * th * th * th * 1.0 / 720.0);

} else { B = (1.0 - cos(th)) / th * th; }

7 A Story

7.1 Wednesday → Friday, Three Thousand More Lines

7.2 Friday Night, Integration Time, Something is Wrong

7.2.1 Can you finish by Monday?

8 A Story

x/yz

8.0.1 “the manuscript-submission instructions for the Physical Review journals

8.0.2 state that multiplication is of higher precedence than division with a slash,

8.0.3 and this is also the convention observed in prominent physics textbooks such

8.0.4 as the Course of Theoretical Physics by Landau and Lifshitz and the

8.0.5 Feynman Lectures on Physics.”

8.0.6 What does it mean?

8.0.7 “the manuscript-submission instructions for the Physical Review journals ...

8.1 But C / C++ / Python / MATLAB / etc. say

$x/yz = xz/y$

8.2 Unit Testing would have caught this Tuesday

8.3 Putting off testing to Friday requires us to debug / search

8.3.1 Actually, this bug was also lurking in MATLAB code transcribed from the same source

9 The Tradeoff

9.1 If you don’t need predictable schedule and high assurance

9.1.1 You can afford the cost of mistakes

9.1.2 You can afford occasional missed deadlines

1. You can afford unbounded debugging time

- Drive Like Junior Johnson

9.2 Otherwise, you need predictable schedule or high assurance

9.2.1 You can't afford mistakes (aviation, CPUs, OSs, platform games)

9.2.2 You can't afford missed deadlines (contracts, FAA, law suits, jail)

9.2.3 You can't afford debugging time (channels are backing up)

- Dig a Tunnel: Test-Driven Development (TDD)

10 The Tradeoff

10.1 TDD trades $O(N)$ dev time for $O(N \log N)$ debugging time

10.2 Debugging is SEARCH

10.2.1 The more you write before you test...

1. the bigger your search space
2. time is unpredictable
 - (a) but only logarithmically if you're good

10.3 TDD is LINEAR

10.3.1 Tests = Specs = Docs \leq Assured Code all at once

10.3.2 Certification = formalized traceability

1. Req'ts \rightarrow Designs \rightarrow Tests

10.3.3 Required by FAA etc.

11 Unit Tests vs. PBT (Property-Based Testing)

11.1 Unit Tests

11.1.1 based on examples “points” invented by programmers

11.1.2 limited by the ability of programmers to invent examples that exercise everything pertinent to a spec.

11.2 Property-Based Testing (PBT)

11.2.1 AKA QuickCheck, hypothesis (Python), rapidcheck (C++), test.check (Clojure)

11.2.2 generates broader tests, statistically sampling the input domains

11.2.3 checks assertions about properties of outputs

11.2.4 best for comparing independent alternative implementations

11.2.5 limited by the ability of programmers to understand broader, non-obvious properties

12 Examples:

12.1 Kalman Filter in C / C++ / Python

12.2 An Interview Question

12.3 Time Warp Operating System

13 Abstract:

13.1 Test-Driven Development (TDD) can deliver higher-quality results with documentation

13.2 and more predictable schedules than can slam-coding and debugging, but it can take longer.

13.3 It’s like digging a tunnel to deliver the moonshine versus driving at 100 mph through

13.4 the woods at night with the headlights off. We present tradeoff analysis and examples

13.5 from embedded aviation code in C / C++, where certification authorities often require

13.6 traceable documentation and high assurance (machine-checked coding standards; proofs;

13.7 exhaustive or statistical unit testing; more). We also present examples in Clojure, which

10

13.8 facilitates lightweight formal specs and generative testing for JVM code.

13.9 Normal unit testing is based on point-like examples invented by programmers (one point

13.10 in input domain mapped to one point in the output range). It’s limited by the ability of

13.11 programmers to invent examples that exercise everything pertinent to a