An Association for All IT Architects

Value (and Costs) of Custom Software Development

IT ARCHITECTURE SUMMIT, 23 AUGUST 2017

Brian Loomis | Dewpoint

Working agenda:

- Architects must understand sources of value and sources of “bad” costs
- Architects must engage to increase value and reduce cost throughout the lifecycle
- Development organizations should invest in getting better over time
- Talk about value then costs

What’s the first thing we do on a project?

Dewpoint

Value of software

- Defined in economic or business terms, not in SLOC or effort
- Post-internet boom, this can be defined as

  # users * average return per user

- Decreases over time as technology becomes mainstream in a domain
- Return per user can be capped by delivery market
  - E.g., an embedded device or IoT can only be less than the whole device pric
  - Discretionary software is bounded by advertisement rates or discretionary income
  - B2B software (service provider) is limited by business model dynamics

Dewpoint

Economic value

- Requirements-based approach fixes the effectiveness
- “The experienced computer manager who knows (if only intuitively) what equipment can be obtained at various costs and how it can best be utilized *may* well be acting rationally when he proposes a set of specifications and accepts the lowest bid consistent with his requirements…”
- The need to look over multiple iterations or across the range of effectiveness options is required to get most cost effective solution (marginal increases go to zero)
- “The concept of a requirement or need is completely foreign to an economist”

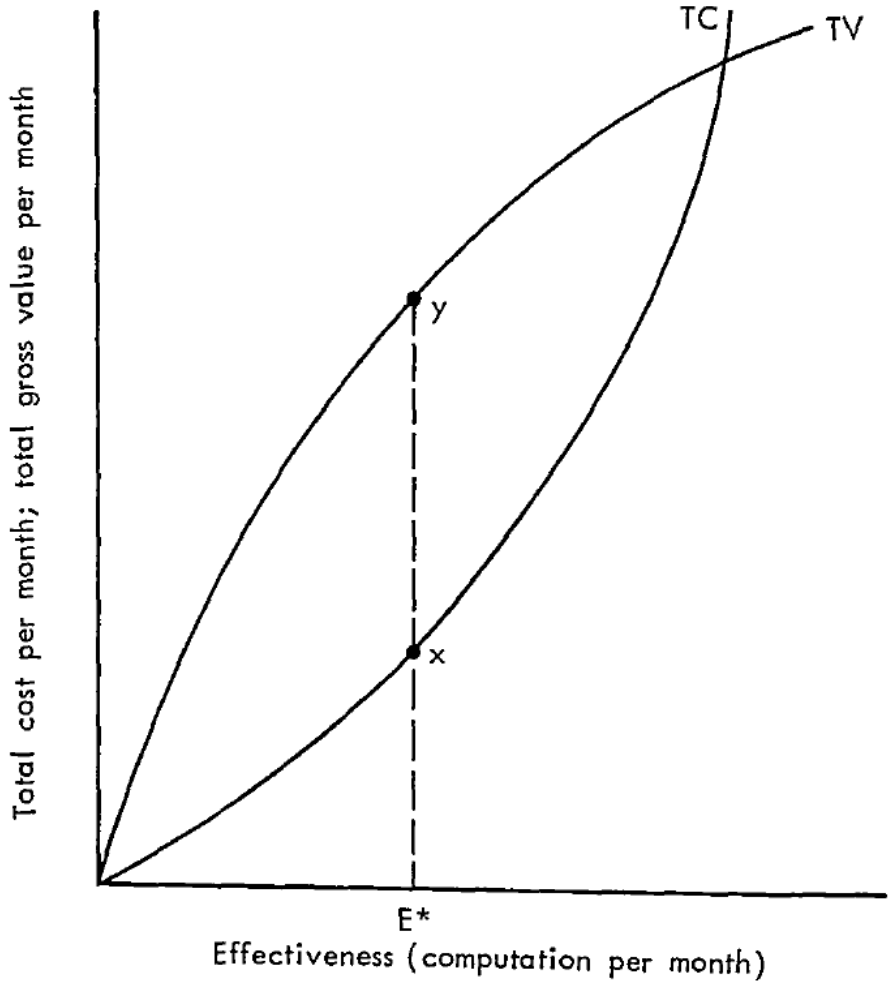
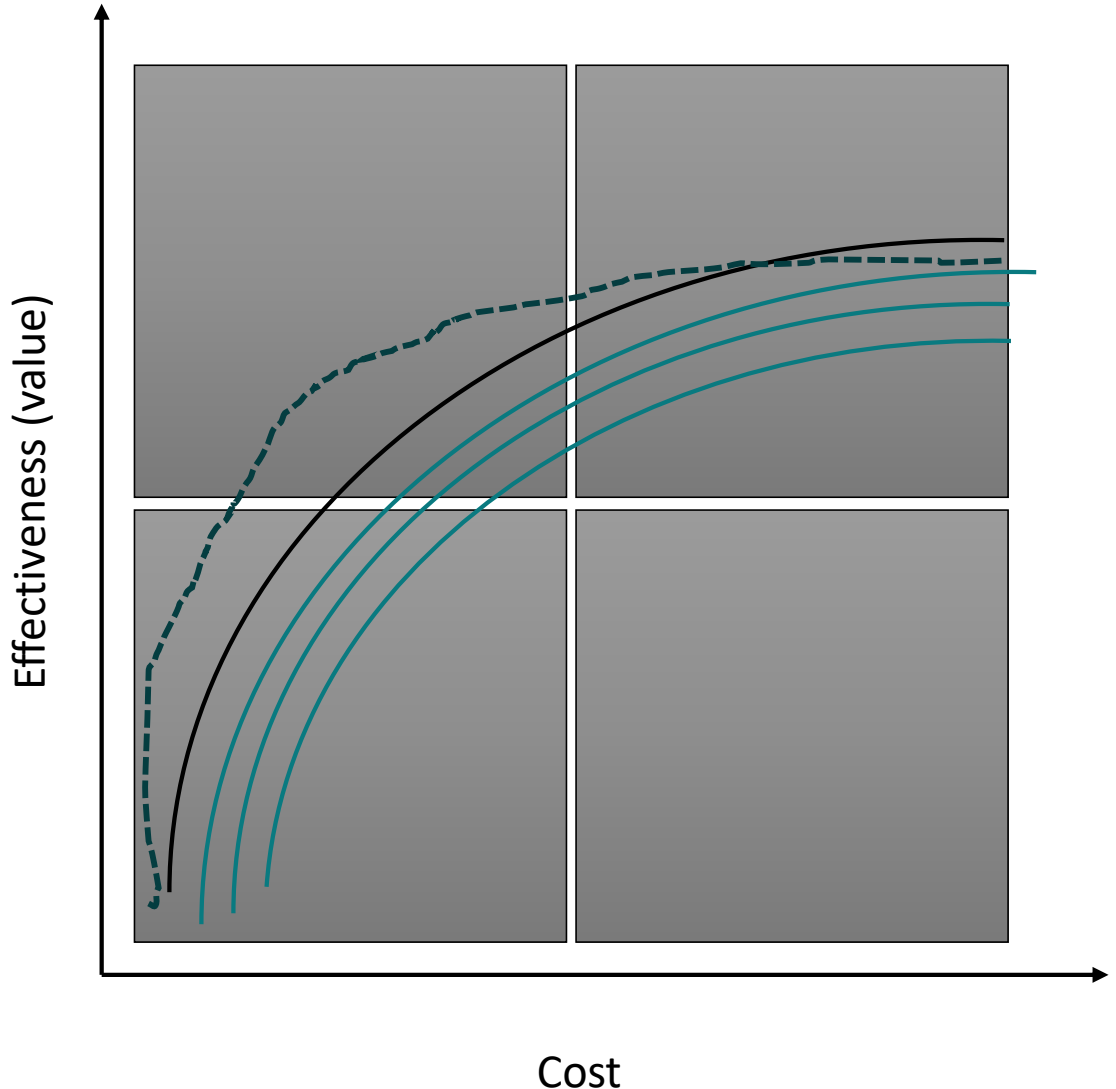


FIGURE 1–4. Maximizing net value.

Dewpoint

Cost curves vary based on experience, technology



Differing cost curves through different implementations – some may yield different points of maximum efficiency

Dewpoint

Increasing value

- Use the BMC to understand the value in quantitative terms and avoid “fuzzy” markets
  - For IoT or cloud systems, the value is not infinite
- Understand what you will build and what you will buy as consumable subsystems (can also point to pivot areas)
  - Digital transformation is often in the custom software domain but it will become commoditized
  - Tools will increase your flexibility (lock-in overrated)
- Refactoring is absolutely critical to anticipating demand
- Define axes of flexibility
- Factor in speed to solution – do the MVP things first, not the stuff you’ll rely on libraries to do later (buying capability)
- Plan for multiple “real” releases
- Software is (initially) a low cost to acquire nowadays – economically, in great supply - so that we do not scrutinize costs or even think of them in a resource-constrained way: i.e., we spend a lot of $$ on marginal yield/efficient software and accept this inefficiency
  - Direct business revenue (attributable) preferred over internal cost center for custom software (much more likely to be successful if we have higher value prop and hence more tolerant budget)

- **Fallacy #1:** making software for a single customer and then grow to bigger market will get me the #s (a.k.a. the second customer is pure profit) → it actually depends on how generic you make the software and process it supports

Dewpoint

Cost of software

- Adding up the (predominantly) labor and other resources required to get custom software to market
- If outsourced, there may be anomalies in cost due to sparse suppliers (locally) or sparse buyers
- Cost is often paid up front for development, so start time matters (if value decreases over time)
- Usually a risk premium involved in estimate

90/90 Rule: The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time. — *Tom Cargill, Bell Labs*

Dewpoint

Parametric models are dead… Long live parametric models!

- COCOMO as the first broad study, models like SEER, SLIM afterwards, use function points, use case points or SLOC
  Can one developer, part of a tester, part of a PM do a two-page spec in a month?
- Group “planning poker” when truly independent, maybe on Fibonacci sequences…

- Better than the alternative…
  - Most projects are “expert estimated”, some are even estimated to available budget, I hear
  - Most projects suffer the **planning fallacy**: optimistic estimates of things we may not have done before (known-unknowns and unknown-unknowns) based on experience which is often “corrected”
  - Technical complexity, size of spec, environmental complexity play a role

- The goal is not the specific model coefficients but that we keep a database of comparable results and make better projections; corollary: models don’t help if you don’t use them!
- Architects need to not just understand but convey the limitations of the iron triangle if PMs cannot

Dewpoint

How to change procurement of software

Customers want predictable cost (and more importantly predictable value)

- Once cost is fixed, the amount of software which can be delivered is fixed as well (you just may not know it!)
- The customer could be brought more formally into prioritizing what gets done first (with the explicit assumption that we can't do everything)
- What mechanism can we put in place to get trued up on cost? How honest are you with your customer? The buyer of the software also has to have management reserve and agree that outsourcing involves profitability. Change orders? Cost or profit sharing?
- Architects *can* do things later in the project to reduce cost
- Agile does not really help us here

Dewpoint

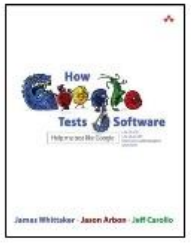
If bugs are 30% of cost…

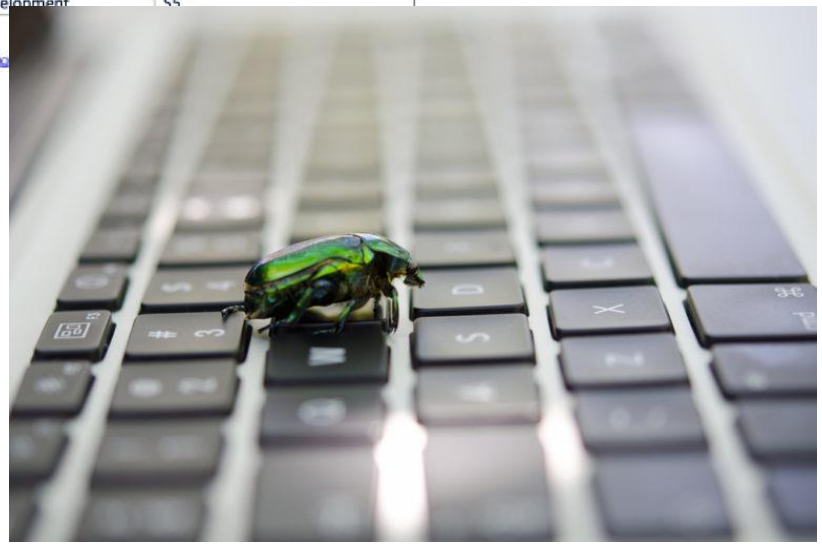
Half your time is spent fixing bugs

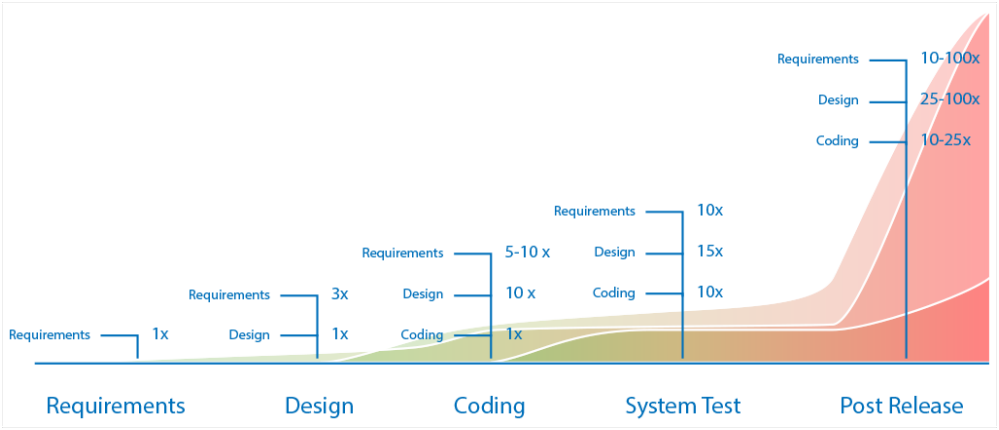
1.	Source Lines of Code (KSLOC) Generated Per Year		200
2.	Average Bugs Per 1000 SLOC[ii]	x	8
3.	Number of Bugs in Code	=	1,600
4.	Average Cost to Fix a Bug[iii]	x	$1,500
5.	Total Yearly Cost of Bug Fixing	=	$2,400,000
6.	Yearly Cost of an Engineer[iv]	/	$150,000
7.	Number of Engineers Consumed with Bug Fixing	=	16
8.	Engineering Team Size	/	40
9.	Percentage of Staff Used for Bug Fixing	=	40%

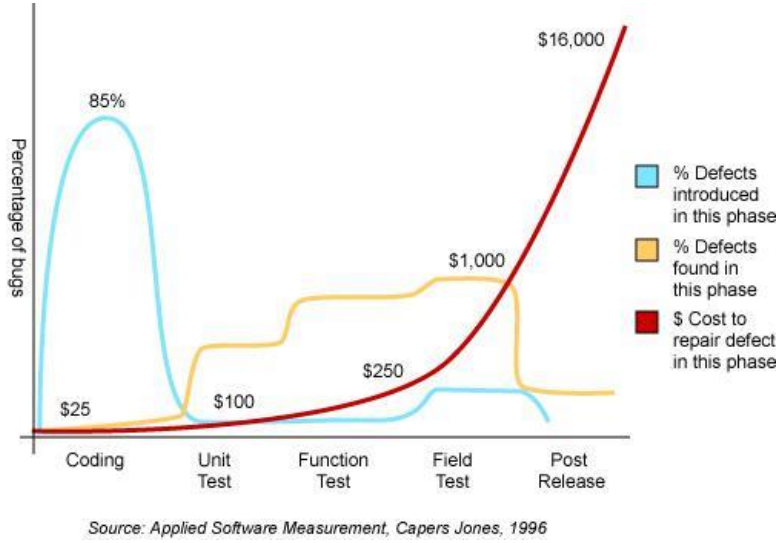
Software Testing Phase Where Bug Found	Estimated Cost per Bug
System Test	$5000
Integration Test	$500
Full Build	$50
Unit Test / Test Driven Development	$5

http://www.newelectronics.co.uk/article-mag





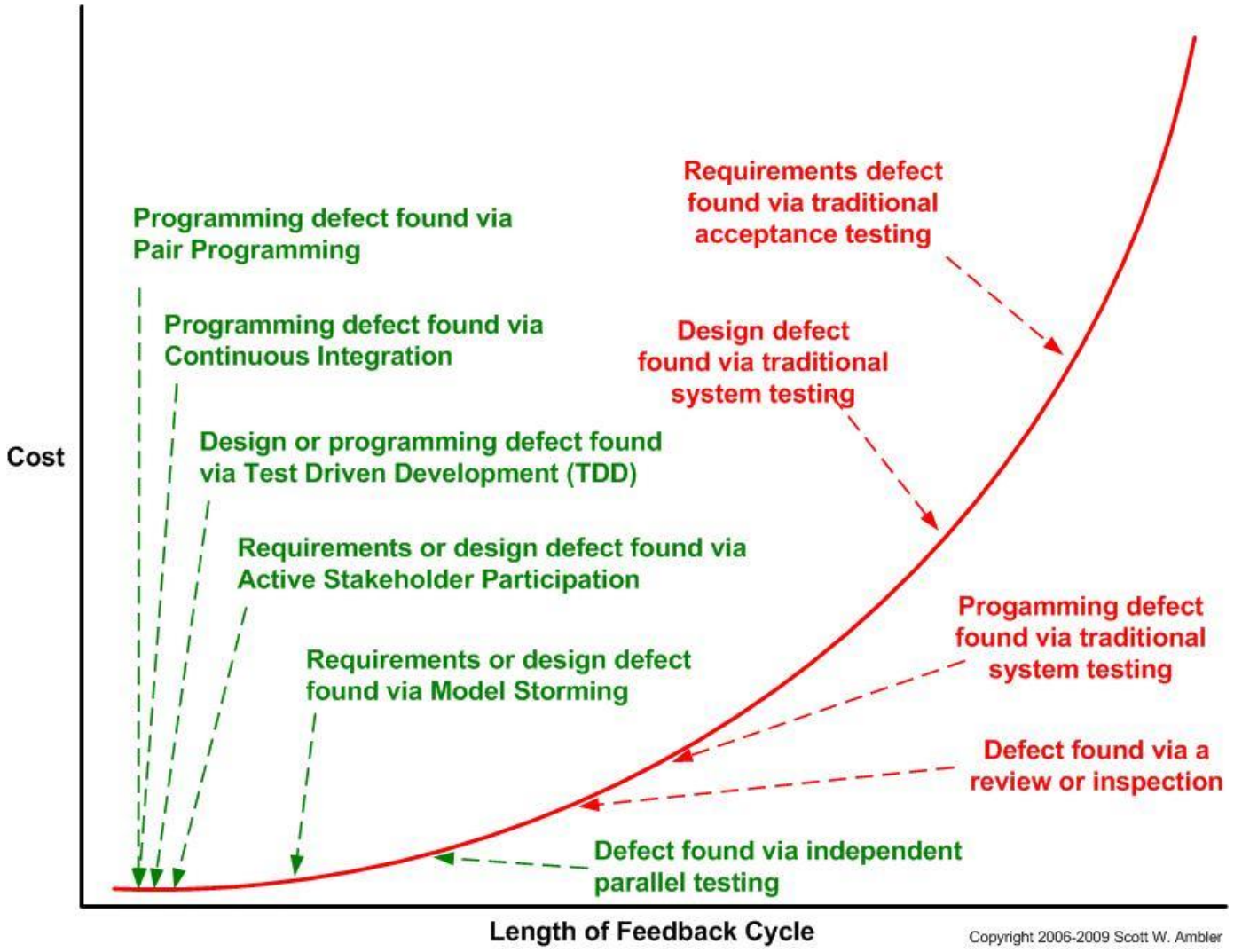




Source: Applied Software Measurement, Capers Jones, 1996

Dewpoint

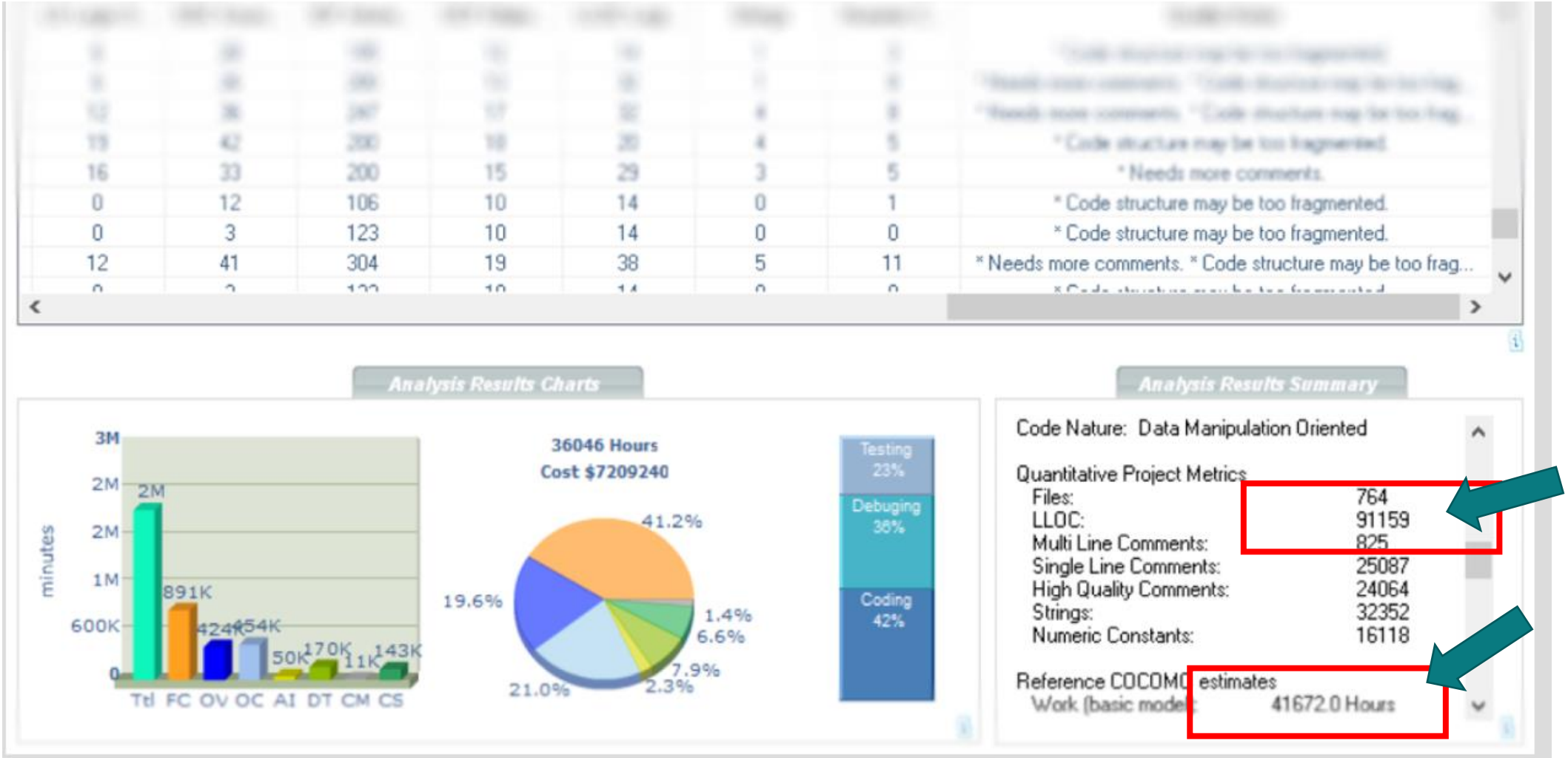
An agile perspective



Dewpoint

Running COCOMO tools…

16	33	200	15	29	3	5	* Needs more comments.
0	12	106	10	14	0	1	* Code structure may be too fragmented.
0	3	123	10	14	0	0	* Code structure may be too fragmented.
12	41	304	19	38	5	11	* Needs more comments. * Code structure may be too frag...



Not far off

Closer to 5K hours

Dewpoint

How architects can reduce costs

- Spend more time in requirements on the right activities (MVP activities)
- Set the patterns – not just GoF, but error handling, database persistence, queuing, message passing, secure coding, how to test…
- Less doco, teams with longevity together, refactoring – agile
- Use tools, justify moving away from tools
- Determine the whole platform (COTS inclusion for basics & platform components)
- Really question one-off late requirements and “process descriptions” of how it works today; avoid heavy integrations
- Pair programming or getting team members to work in different parts of code
- Describe bugs accurately – often in the rush to be responsive, we stop doing this…
- Fix the whole bug

- **Fallacy #2**: I can skip testing because my team writes better code than average
- **Fallacy #3**: I don’t need patterns and can use junior developers to get the rate structure down

Dewpoint

Thank you!

Brian Loomis, CITA-P

bloomis@dewpoint.com

Dewpoint