**A Position On Domain Driven Design DDD**

Based on <https://domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf>

And other internet sources like 2 utube lectures by Eric Evans.

Version 29 March 2019

By

[tom@Gilb.com](mailto:tom@Gilb.com)

Some people I met at a recent meetup in Oslo asked my opinion on DDD, so this is an answer.

1. I was planning to write a paper comparing it with other methods, but I concluded it was uninteresting to do, & not worth my effort.

2. It is clearly a method limited to modelling software (programs). I find this uninterestingly narrow. There are plenty such methods, and I hope the programmers of the world are happy with it. I am personally only interested in languages for modelling any and all **system** domains. (Like Planguage: as in CE book <https://www.gilb.com/p/competitive-engineering>)

3. The DDD descriptions are full of words and sentences which I find little useful meaning or insight into. Words words words. Not very long on examples. The Youtube with an example was toothless and boring. Why? It is not at all clear what the advantage, or point of it all, is.

* There are no assertions of the overall, or the detailed, advantages.
* **There are no measures, or even hints of measures, or research indicating what the advantage or point of it all is: like reduced time, ease of maintenance, reliability, quality control - at the level of the model.**

**4.**  There are many software and system methods that do have assertions about their attributes. Many have actual real case studies. Some do have real measures of goodness, and do serve the entire system, not just software. My own books/Papers/slides at [gilb.com](http://gilb.com) have lots of examples of this, so I believe in evidence-based decision-making. I am in favor of things that others have measured, and which I can measure in a project of company.

5. I believe in science and rational thinking, not software religions.

6. Here are some simple examples:

1. J. Terzakis,

"The impact of requirements on software quality across three product generations,"

2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, 2013, pp. 284-289.

https://www.thinkmind.org/download.php?articleid=iccgi\_2013\_3\_10\_10012

Which is based on years of research on real Intel projects using my Planguage and SQC methods, which have been in use by over 20,000 engineers for over 20 years at Intel alone.

Highlights are: 233% greater engineering productivity and 50x fewer requirement defects than initially submitted.

These are requirements for the embedded software in Intel products.

2. IBM Cleanroom:

MIlls and Quinnan Slides

<http://concepts.gilb.com/dl896>

Mills, H. 1980. The management of software engineering: part 1: principles of software engineering. IBM Systems Journal 19, issue 4 (Dec.):414-420.

Direct Copy

<http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan>

This is a set of software methods known collectively as ‘Cleanroom’. The simple bottom line is years of **on time**, **under budget** delivery of **fixed price** (lowest bidder wins), **fixed deadline** NASA and DOD contracts, at the **highest levels of quality and complexity** (like Space Shuttle).

This is by the way a great Agile (as it should be) method, but it is quantitative value engineering way beyond popular agile (40% to 63% total project failure) and Scrum ‘only 19% project failure (Sutherland).

Cleanroom is pretty much identical to my Evo method (see the CE book above) which has been validated (by public written reports of measured experiences) by Intel and HP among many others. But I like to cite the IBM experience.

NOW, WITH LOTS OF METHODS LIKE THESE FOR SOFTWARE, why would anyone have any interest in DDD. The answer is: the same uncritical people who believe in Scrum. The ignorant masses who like to fail. We need to move from high failure rates to near zero failures of software.

Enough of my opinions.

Feel free to argue, or point to argument about, any measured useful attributes of DDD to me, but I could not even find a hint, or even a false unfounded claim for such advantage. I will listen to evidence based rational argument. But at the moment there is none.

Tom Gilb

[tom@Gilb.com](mailto:tom@Gilb.com)

HERE IS AN EXAMPLE OF SOME CLAIMS FOR DDD, which are not at all substantiated by any evidence (facts, measures, studies).

SOURCE: https://www.dagdoni.com/blog/2018/4/22/a-short-introduction-to-domain-driven-design

“We also want to have a design system that can scale and guide us to create the right domain entities. This will ensure that our business and software are aligned, well documented and understood by everyone. That’s were domain drive design can help.

You want to improve your craft and increase success on your current project. You care deeply about crafting software that correctly models your business needs and also performs at scale anywhere using the best software architecture. This article will inspire you to use DDD in your project.

What is DDD?

Domain Driven Design is a set of tools that help you in designing and implementing software that delivers high value both strategically and tactically.

LET ME PARSE THIS LARGE SET OF UNSUBSTANTIATED CLAIMS; TOM GILB

Domain Driven Design is a set of tools that help you in designing and implementing software that delivers high value both strategically and tactically.

1. scale and

2. guide us to create the right domain entities.

3. This will ensure that our business and software are aligned,

4. well documented and

5. understood by everyone. That’s where domain drive design can help.

6. You want to improve your craft

7. and increase success on your current project.

8. You care deeply about crafting software that correctly models your business needs

9. and also performs at scale anywhere using the best software architecture. This article will inspire you to use DDD in your project.

What is DDD?

Domain Driven Design is a set of tools that

10. helps you in designing and

11. (helps you) implementing software

12. that delivers high value both strategically

13. and tactically.

IN SHORT

Nice words, not defined = no clear meaning.