IDA

Software Speed Limits

What controls how fast we can modernize?

David Tate and John Bailey

May 2020

We have a need for speed

Everyone ☆ agrees that

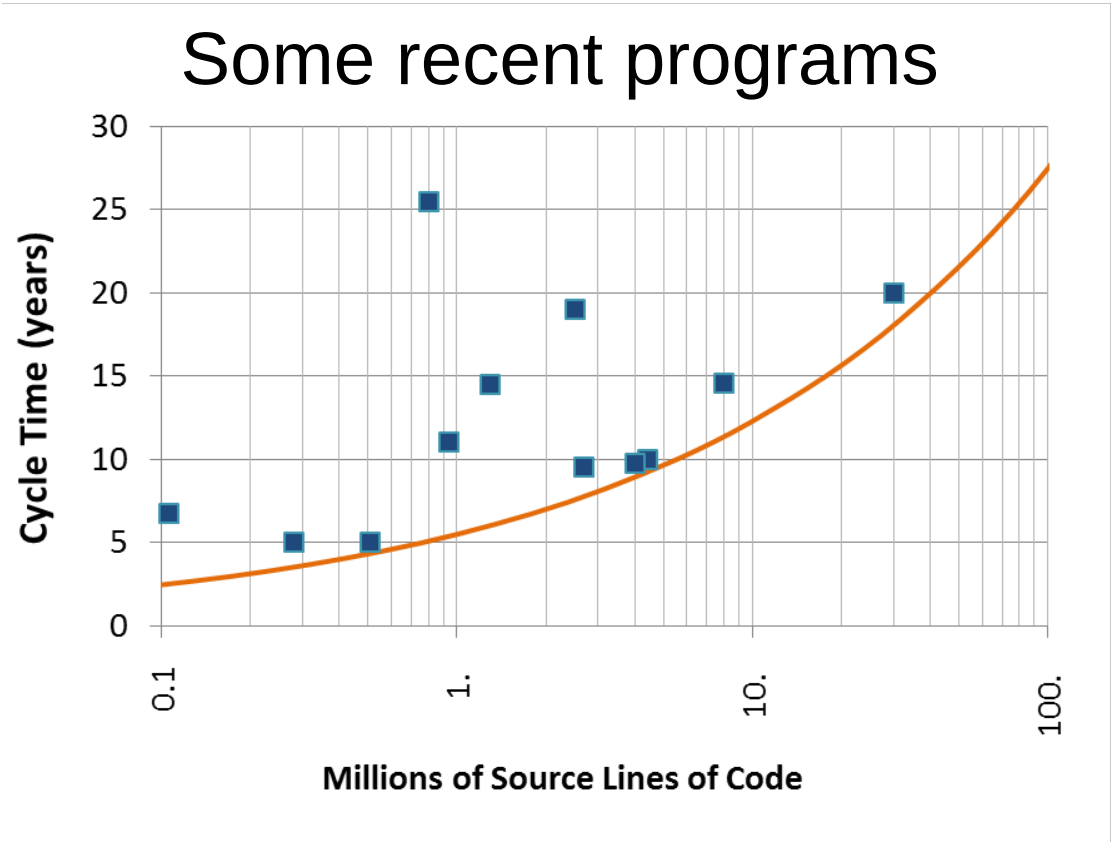
1. DoD needs to field new capabilities **faster**
2. Future capabilities will be **software-enabled**

☆ SecDef, DepSecDef, USD(A&S), Defense Science Board, Defense Innovation Board, National Defense Strategy, …

Software is now the limiting factor in speed

Increasingly, the time to develop the software is on the critical path for system development

This has been true of space and C4 systems for decades; it's now true of nearly all systems



So how fast can software development go?

We examined the factors that constrain speed of software development – in particular, software associated with new fielded capabilities in mission systems.

In rough order of importance, they are:

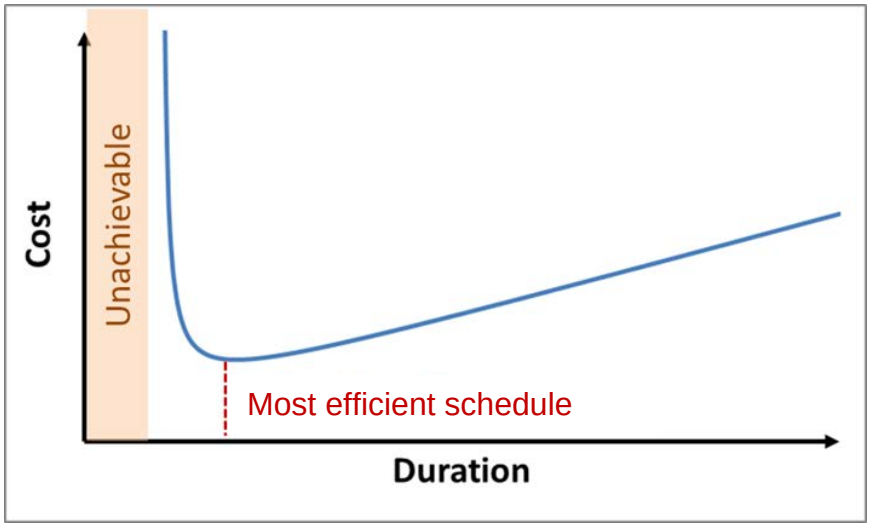
1. Required functionality
2. Architecture
3. Technology maturity
4. Resources
5. Testing strategy
6. Contract structure
7. Change management

1. Required functionality

What will the system do?

For a given set of required capabilities (and specified architecture), there is a "most efficient" schedule to meet those requirements with software.

Attempts to go faster than this incur extra cost or cause the project to fail.



Agile development does not change this

Within each phase of an agile development project, the same relationship between content and efficient schedule still holds. Agile makes projects faster by:

1. Dropping or deferring requirements
2. Improved testing strategy (which we will return to)
3. Faster stakeholder feedback

Corollary: if the Minimum Viable Product is complicated, getting to the point where you can start sprinting might take a long time.

(What is the MVP for a GPS ground control system?)

2. Architecture

On my car, it is easier to change the tires than the doors

That was a deliberate design choice by the auto maker

For software to be easy to modify in the future, similar up-front choices are necessary

Architecture time and effort depends on future needs

The architecture is part of the MVP

The time and effort needed for architecture and design depend on the answers to questions like…

- What are the cybersecurity needs of the system?
- How important is it for the system to be upgradeable?
- How fast/frequent/extensive do those upgrades need to be?
- What is the intended lifespan of the software?
- How portable/reusable does the software need to be?
- What other systems will it need to interface with? Will those systems also be changing over time?

3. Technology maturity

Things go faster when you already know what to do

Technology Readiness Assessments tend to focus on hardware - critical technologies

Critical software technologies can also be immature – e.g., machine learning and autonomy

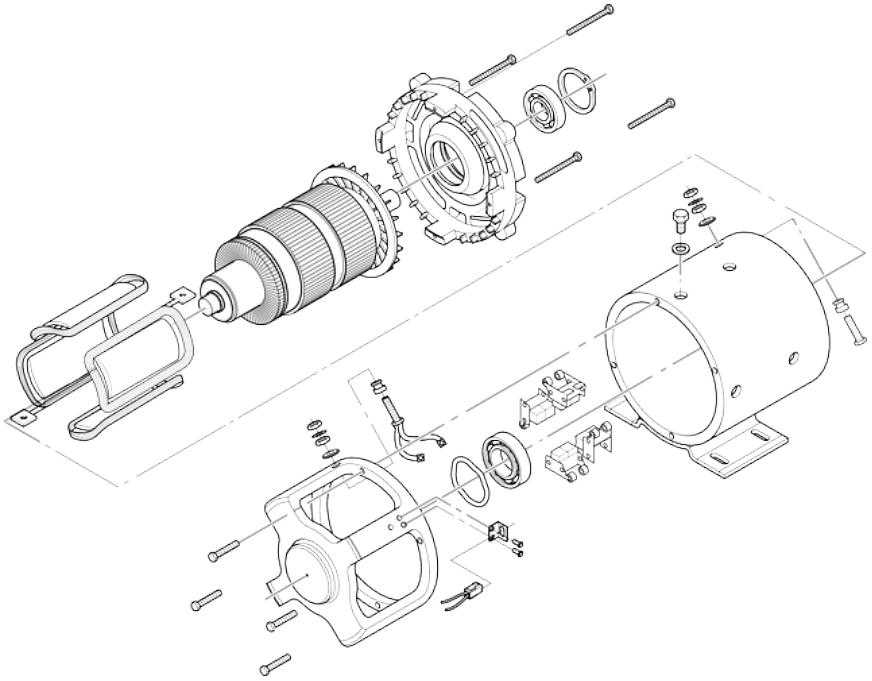


Image source: https://www.needpix.com/photo/download/22775/diagram-assembly-parts-instructions-labeled-mechanical-assemble-sketch-free-vector-graphics

Immature hardware can also affect software

Once past Critical Design Review, it is easier to make major changes in software designs than hardware designs

As a result, immature hardware technology can cause significant additional software effort, including re -work

4. Resources

Available resources constrain the execution of any project

For software, the vital resources are skilled labor, development infrastructure, and (increasingly) sufficient data

…and of course, $$$

An adequate supply of the right skills is not a given

There is increasing evidence that the supply of cleared software professionals is not keeping up with DoD demand

Staffing shortfalls, on-the-job training, and mismatches between funding streams and staffing needs all drive delays in project execution

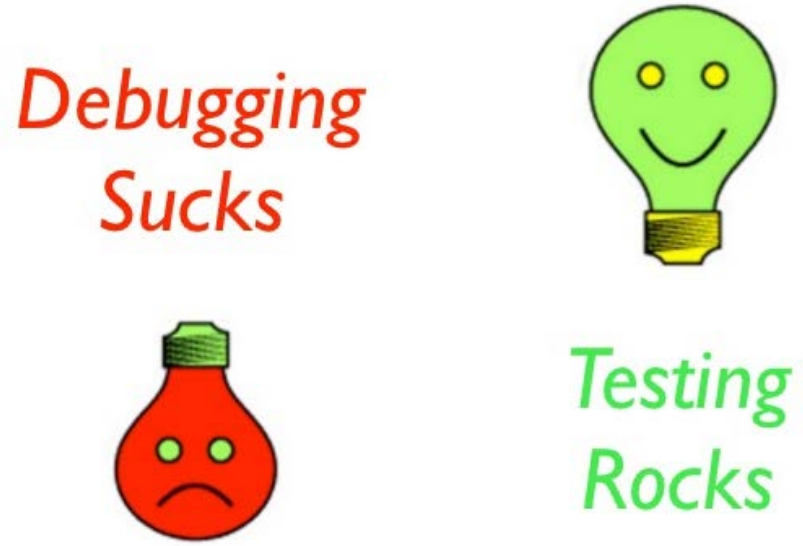
These issues do not stop at delivery – development effort continues for as long as the software is used, and requires many of the same skills as initial development

5. Testing strategy

It is hard to do too much software testing, but it is easy to do testing wrong – or too late

Finding and correcting defects is the largest identified driver of cost and delay

Modern software development practice prioritizes early testing to avoid later debugging



Testing is free

When done right, the cost of testing is negative – the more (and earlier) you do it, the less you spend on development

Agile development assumes a culture of continuous stringent testing from Day One

DoD programs, who have to pay for testing out of pocket, do not always share this philosophy

6. Contract structure

If you buy a car that can only be serviced by the dealer who sold it to you, you can expect to pay a lot (and wait a lot) for maintenance

The same is true for software

Image “Tesla car PNG” from website http://pngimg.com/download/62082, reused under Creative Commons 4.0.

Contractors will not give up monopolies for free

DoD would prefer that all software be modular, reusable, and fully open to enhancement or replacement by competing third-party vendors

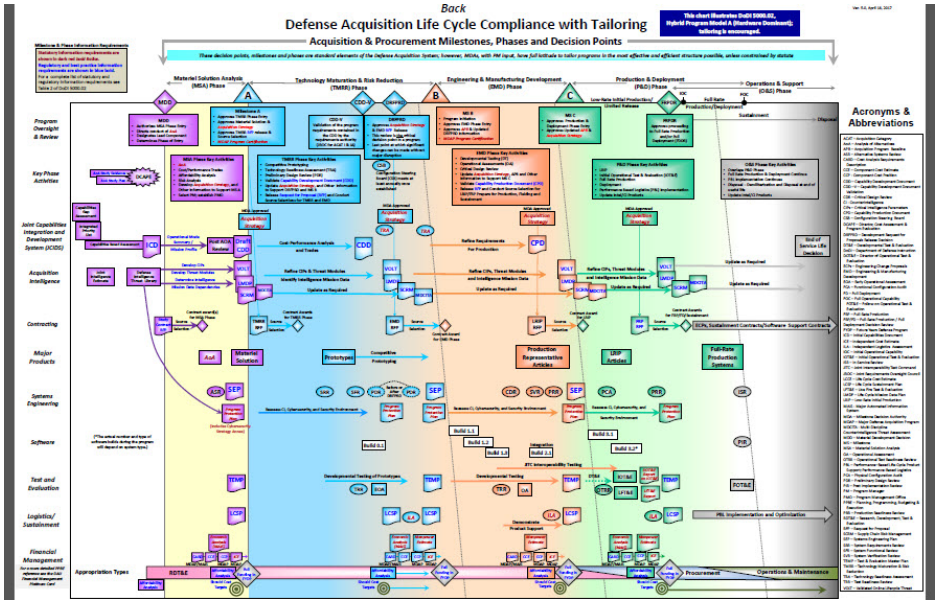
Contractors would prefer to have a monopoly on future upgrades, which can be significantly more lucrative than the original development contract

DoD struggles to craft contract terms and incentives that permit both DoD and contractors to be satisfied

7. Change management

In theory, the acquisition process is supposed to go like this:

1. Characterize the future fight and define mission needs.
2. Identify capability gaps and identify alternative ways to mitigate them.
3. Analyze the alternatives and select a preferred alternative.
4. Set threshold requirements.
5. Develop and field a system that meets those requirements.
6. Maintain that system.



The days of static procurement are long gone

We don’t buy the thing we said we were going to buy

The thing we are buying changes constantly

Sometimes it’s a service, not a thing (e.g., space launch)

We don’t consistently distinguish between new programs versus new products within existing programs

Our planning and oversight aren't made for this

They assume a static design and known quantity

Honesty about future upgrade plans is punished

Improved capability is punished (e.g., Nunn -McCurdy)

Agile development provides a politically acceptable excuse to openly do what we've already been doing, but faster and more efficiently – if the requirements stakeholders will permit it

Software takes time for valid reasons

How much time the software will take is determined by what we want it to do over its life cycle and the environment we'll be inserting it into

To go really fast in the future, we will need to have made certain choices in the past – including some choices that look slow at initial development time

Architectures and data rights will be just as important as acquisition pathways or development philosophies

We should probably be worried about the industrial base


IDA