DCG
david consulting group
TRUSTED ADVISOR

"How can we optimize our SDLC to maximize demonstrable value to the business?"

March 2015

Scope of this Report

This report investigates how changes to the SDLC (Software Development Life Cycle) can improve the delivery of demonstrable value to the business. We consider how we might measure "demonstrable value" in a way that the business will understand. We review the theory of "Lean Software Engineering" and we suggest some ways that the theory can be applied to optimize different SDLC's. Finally, we discuss the importance of Value Visualization – requiring each story or requirement in the SDLC to have a demonstrable and highly visible set of business value criteria to drive tactical decision making.

What is "Demonstrable Value to the Business"?

Basically, most software development organizations are driven by demands (or possibly polite requests) from the business(es) that fund the software development. This is not all the work they do because some work is self-generated either from the software development group or the rest of IT but, generally, this second category of work still has to be accepted for funding by the business and prioritized against business needs.

So, "Demonstrable Value to the Business" could be simply delivering to the business what it asks and doing so in accordance with the "iron triangle" of in scope, on time and on budget. In "*The Business Value of IT*," Harris, Herron and Iwanicki argued that "Business Value" tends to be in the "eye of the beholder". While this is an important ingredient of the definition, it is not sufficient. Some rigour must be applied beyond simple customer satisfaction because, at the end of the day, the success of an organization will almost always be judged in monetary terms. Even non-profits must be able to stick within the available budget while delighting customers.

Inevitably, the best way to introduce objectivity into a business value discussion is to follow the money, however difficult and apparently unjustified this may seem to the participants. Certainly, there are value types that cannot be measured in dollars and cents but we would argue that such situations are relatively rare in the business of software development. Hence, while we would always include customer satisfaction when assessing business value, we believe that "demonstrable value" requires the objectivity of financial metrics.

Lean Software Engineering

In our November 2014 DCG Trusted Advisor report, we investigated the meaning of the term “Lean Software Engineering.” That report is a good starting point and recommended reading for this report. To summarize the key points of relevance to this report:

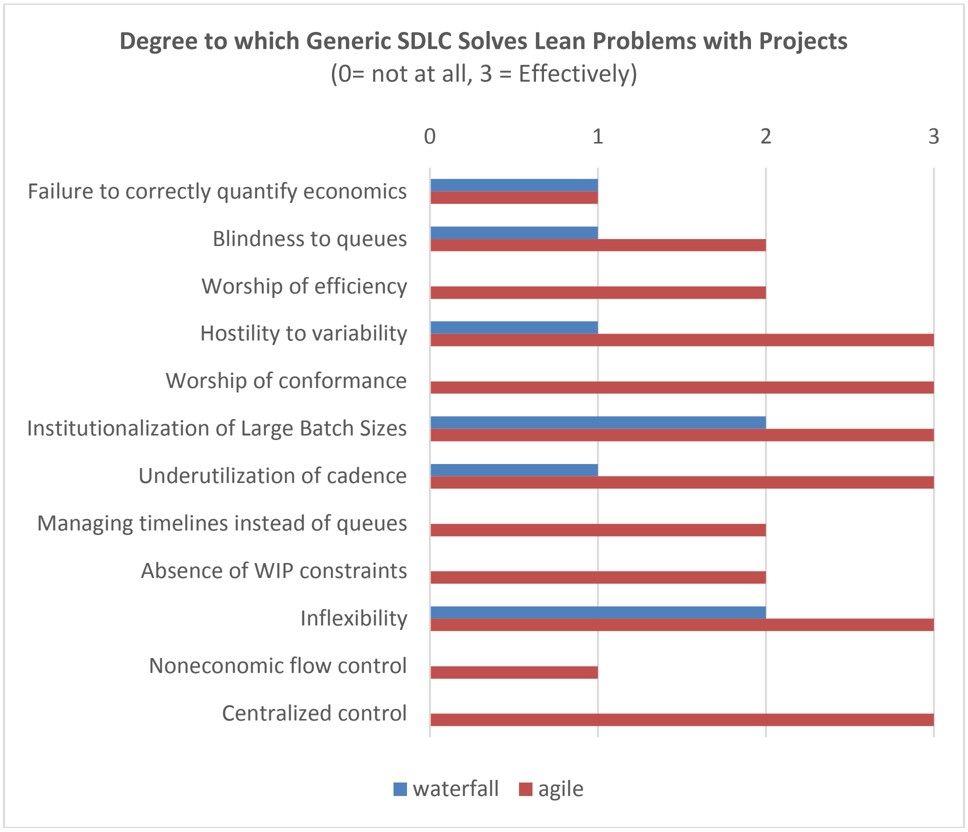
- The Poppendiecks (Mary and Tom) have proposed seven principles of Lean Software Development:
    - Eliminate Waste
    - Build Quality In
    - Create Knowledge
    - Defer Commitment
    - Deliver Fast
    - Respect People
    - Optimize the Whole
- Lean software engineering is not an SDLC but an optimization philosophy that can be applied to all SDLC’s. The following practices are often associated with implementations of the Lean philosophy in software engineering:
    - Visual Controls including Kanban Boards
    - Cumulative Flow Diagrams
    - Virtual Kanban Systems
    - Small batch sizes
    - Automation
    - Kaizen (or continuous improvement) Events
    - Daily Standup meetings
    - Retrospectives
    - Operations Reviews.
- Lean principles and practices are applicable to waterfall SDLC’s and embodied in Agile SDLC’s although in neither case is there usually 100% compliance.

In considering lean product development flow, Don Reinertsen identifies twelve problems with traditional product development. Reinertsen was referring to traditional waterfall implementations of product development but, as Figure 1 shows, some of these have been addressed by typical Agile implementations but some are only implied. For example, “Absence of WIP constraints”: Few Scrum implementations has specific WIP constraints but constraining team size and constraining sprint duration implies a WIP constraint.

Opportunities for Optimization

Of course, there are many variations of the generic SDLC models which arise from local customization to address either the problems we have described or other local issues. Hence, for example, our scores here might be modified (up or down!) by an Agile implementation that is not text book Scrum but some combination of Lean/Scrum/XP. That same is true for modified versions of waterfall which, for example, have a strong cadence achieved by pipelining requirements/design, development and testing into, say, three month chunks that endlessly repeat. For this reason, in Appendix A, we have added some commentary to the simple scores of Figure 1.

Figure 1: Problems with traditional approaches to Product Development (after Reinertsen) and the degree to which they are addressed today by typical Waterfall (Blue) and Agile (Red) SDLCs. A Reinertsen problem that has been fully-addressed in an SDLC would score a '3'.



So with the stipulation that the actual SDLC model you are looking at might not score exactly the same way as we have suggested in Figure 1, the greatest opportunities for optimization leap off of the chart:

- For both SDLC's
    - Quantify the economics and make them visible at all levels so that they can be used for tactical decisions
    - Make queues explicitly visible at all levels
- For waterfall:
    - Implement Agile
      OR
    - Implement a Lean Kanban system or some other system that:
        - Sets Work in Process (WIP) Limits on process steps
        - Sets small batch sizes
        - Decentralizes control by implement work-pull instead of work-push.
    - AND
    - Focus on optimizing end-to-end value throughput instead of focusing on resource utilization that may maximize local productivity at the expense of throughput.

How? The Economic View - Value Visualization

We need to associate a set of economic information with each requirement or story that we want to flow through our SDLC. We propose the minimum set in Figure 2 should be added to each requirement/story in an easily visible (see Figure 3) or electronically accessible way.

Figure 2: The Value Visualization Economic Metrics set for every Requirement/Story

Name	Purpose	Examples
Units	Define the units in which the value will be delivered	• Number of subscribers added • Number of effort hours saved • % increase in sales
Payload	Define that actual number of the units that deployment of this requirement/story is expected to realize	• 25 (subscribers) • 3 (hours per month) • 5 (%)
Size	Usually this will be relative size in story points or function points but it could be simple relative size based on modified Fibonacci	• 100 story points • 2000 story points • 13 (from Fibonacci)
Cost of Delay	Per Reinertsen and others, this may be the most important metric for tactical prioritization. It should be in $ but could be relative (Fibonacci). With size as a proxy for duration, it can be used to calculate a rough Weighted Shortest Job First (WSJF) value for prioritization. Value being equal, highest WSJF is pulled first. Cost of delay may also include penalties for late delivery.	• $2500 fine if not ready by year end • $1000/day for rental of expensive resources • 8 (from Fibonacci)
Unit Value when Deployed	Defines the actual value of the defined units at time of deployment. For example, a cost saving economic case may have been made when the resources hourly rate was $60/hr. That work has since been outsourced and the rate is now $40/hr.	• $35 per new subscriber • $50/hr. • $1000 per % increase
Deployed Value	The product of the payload and the Unit Value When Deployed	• $35 * 25 = $875 • $50 * 3 = $150 • $1000 * 5 = $5000

As we have learned from Kanban boards in Agile, visualization is very powerful for team decision making and so it makes sense to associate the economic value data of Figure 2 with a visual model for which we provide a template in Figure 3 and an example in Figure 4.

Figure 3: A possible Value Visualization Graphic

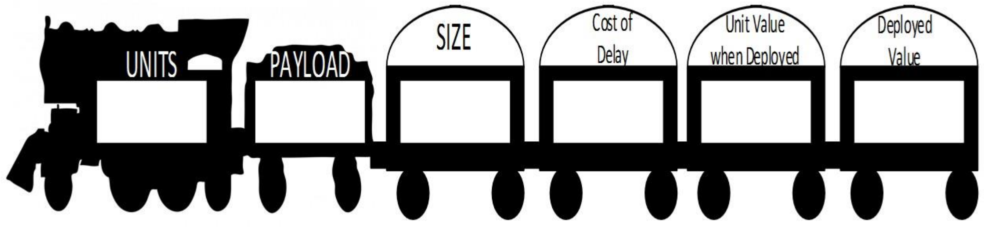
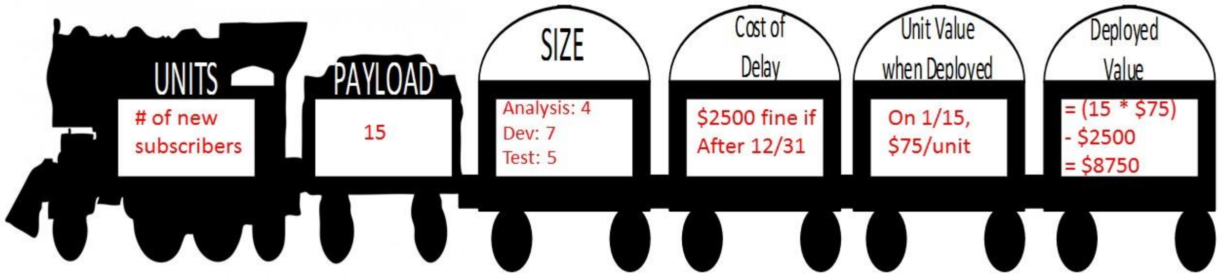


Figure 4: Value Visualization Example1



There are some challenges carrying value visualization data through requirement or story decomposition because the business cases that drive the requirements and stories often map to quite high level requirements and epics rather than the "small batch" level requirements and stories that we want to see flowing through development. We have learned from at least one client that it does not make sense to have economic value data at the lowest requirement/story level because of the difficulty of breaking up the high level economic information into ever smaller units. Hence, at some level of decomposition it will be necessary to stop breaking up the economic data and follow three simple rules:

- Use T-shirts sizes for Value e.g. High, Medium, Low at the lowest levels of story.

- These T-shirt sizes should be inherited from and the same as the lowest level parent requirement/story for which economic data was available.
- Implement a control mechanism to ensure that below the lowest of economic data, all child requirements/stories are connected and prioritized together as much as possible.

There must be an associated process for the third main rule. It is possible that a subset of the stories associated with a particular set of economic data could be deployed before all the stories associated with that data are ready for deployment. For example, in some cases, 90% of the value could be realized by deploying 75% of the child stories. In such cases, decisions need to be made and executed about the value of the remaining "orphaned stories." In short, are they still needed or can they be removed from or deprioritized in the SDLC flow.

The use of Lean-Kanban.

While Agile SDLC's such as Scrum and SAFe are designed to embody many lean principles, waterfall was not originally designed with lean principles in mind. However, that does not mean that moving to Agile is the only course of action available. If there are good reasons for an organization to stick with waterfall for part of its operations then the application of Lean-Kanban principles can help. After all, Lean principles originally emerged in manufacturing environments that tend to be waterfall in nature (a quick reminder/warning that lean manufacturing and lean software engineering are not the same thing as we have discussed in earlier DCG Trusted Advisor reports).

To move from classic waterfall to a Lean-Kanban model, the following minimal steps need to be taken:

- Create a product backlog of requirements/stories in priority order
- [Ideally but not necessarily at first] Add the Value Visualization Data to each requirement/story
- Create a "Ready" step as the first step in the flow to ensure that requirements/stories do not enter the SDLC until they are ready to be worked. Put a WIP limit of less than 10 on the Ready step
- Create a "Ready to Deploy" step as the last step in the flow with no WIP limit
- For each step in the waterfall SDLC (e.g. Analysis, Design, Code, Test), create two sub-steps: "In Progress" and "Done." Each whole step should have a WIP limit for of less than 10.
- Allow the staff in each step to pull requirements from the preceding "Ready" or "Done" step if, and only, if bringing in that requirement/story does not exceed their WIP limit. Staff should pull by highest value or WSJF.
- Use Cumulative flow charts to track and predict bottlenecks and shift resources or WIP limits to optimize value flow.

Conclusion

We are capable of building and running effective Waterfall SDLC's but they are not necessarily efficient in optimizing value flow. Worse, waterfall SDLC's are not good at visualizing the data needed to improve value flow and they tend to be poor at implementing lean principles. Agile SDLC's are much better at implementing Lean principles and so improve value flow but even Agile SDLC's are not optimal if they do not have a way to include some basic economic data in their tactical decision making.

Sources

- “The Business Value of IT,” Harris, Herron and Iwanicki, CRC Press, 2008.
- DCG Trusted Advisor Report, November 2014, “What is meant by the term ‘Lean Software Development,” http://www.davidconsultinggroup.com/insights/publications/ta-archives/lean-software-development/
- “Implementing Lean Software Development – From Concept to Cash,” Mary and Tom Poppendieck, Addison Wesley, 2007. [Or, indeed, any book by Mary and Tom on this topic!]
- “The Principles of Product Development Flow – Second Generation Lean Product Development,” Donald G. Reinertsen, Celeritas Publishing, 2009.
-

Appendix A: Further comments on the scoring in Figure 1

Problem	Addressed in Waterfall SDLC's Today?	Addressed in Agile SDLC's Today?
Failure to correctly quantify economics	Some insight is usually available into budgets, change requests and overruns (because these are so common)	Business priorities are implied by ordering of backlogs but economic data is not usually available at story level
Blindness to queues	Queues are "felt" more than visible in the form of "pressure of work" building up at each stage.	Product Backlogs make queues visible outside sprints and Kanban boards help inside sprints but queue management is implicit not explicit.
Worship of efficiency	Generally maximize productivity by maximizing local utilization	Pull to capacity helps but sometimes local pressure to increase velocity
Hostility to variability	Although designed to exclude change, change request processes exist and are used).	Designed to manage variability at the team level.
Worship of conformance	Guilty as charged. Broadly, a "factory" model for software dev.	At the team level, much room for creativity.
Institutionalization of Large Batch Sizes	Generally acknowledged that big projects are bad but they are still out there.	Small teams and small iterations drive small batch sizes.
Underutilization of cadence	Often "macro" cadence through "pipelining" of stages.	Patchy. Some orgs. Impose fixed iteration lengths others let teams choose.
Managing timelines instead of queues	All timeline management. No queue visibility or management.	Inter-team queues structurally removed but intra-team queues not visible
Absence of WIP constraints	WIP constraints not explicit, worked pushed not pulled	WIP constraints not explicit but work pulled so WIP limits implied
Inflexibility	Flexibility has grown over time through necessity using customization of different workflows and change requests when requirements change	Highly flexible
Noneconomic flow control	All flow control is non-economic. Either FIFO or who shouts loudest.	Flow control is non-economic because teams often pull work without economic data on that work. There is little explicit value data – rather there is business prioritizations which may implies some sort of economic analysis. Good agile does recognize implied value data but bad agile does not
Centralized control	Designed to be centrally controlled although central "control" can be illusory where there is limited visibility.	Agile can be highly decentralized but scaling with full decentralization is a challenge which begs the question, "Is total decentralization actually good?"