Of Hornets and Hippos

In the 1990s, the United Stated military decided to replace three of its aging fighter aircraft models. The F-16 Fighting Falcon was a conventional take-off and landing, single engine fighter jet for the air force. The F-18 Hornet was a fighter for the navy capable of taking off and landing from air craft carriers. And the AV-8B Harrier was a marine aircraft capable of take-off from short runways and landing vertically, like a helicopter.

F-16 Fighting Falcon

F-18 Hornet

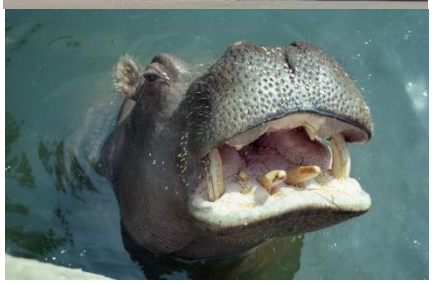
AV-8B Harrier

Military leaders opened a competition, eventually narrowed to Boeing and Lockheed Martin, to replace all three aircraft with a single family of one new fighter. The new plane, dubbed the Joint Strike Fighter, needed to include an air force configuration with conventional take-off and landing, but with a Mach-1 top speed and aerial refueling capability. The navy configuration required carrier take-off and landings. And the marine variant needed to take-off from a short runway with full fuel and weight, and also hover and land vertically at low fuel and weight.

Plus, all configurations were to have 80% of their parts in common with one another. The hope was that, with extensive parts commonality, the plane could be produced less expensively, by employing economies of scale.

In October 2001, military leaders crowned Lockheed Martin's plane the winner. Several colleagues of mine who worked on the program are still convinced that Boeing made the better airplane. But perhaps an undocumented requirement contributed to Lockheed's success: the "cool factor." Lockheed's plane looked more like a traditional fighter, which is to say, it looked like a bird of prey. Boeing's plane, with its unusual air intake below the cockpit, looked a bit like a hippopotamus.

Lockheed's JSF entry



Boeing's JSF entry

Virtually since the competition award in 2001, the JSF has flown into problems. Cost overruns, schedule delays, and quality issues have degraded the program's reputation. A New York Times Magazine article's headline declared it "America's dysfunctional trillion dollar program."

But the same article described efforts to improve the program using agile software development. For acquiring software upgrades to the jet, the author states:

A trial program staffed with a team of Air Force and Lockheed coders proved that the (agile software development) method works.

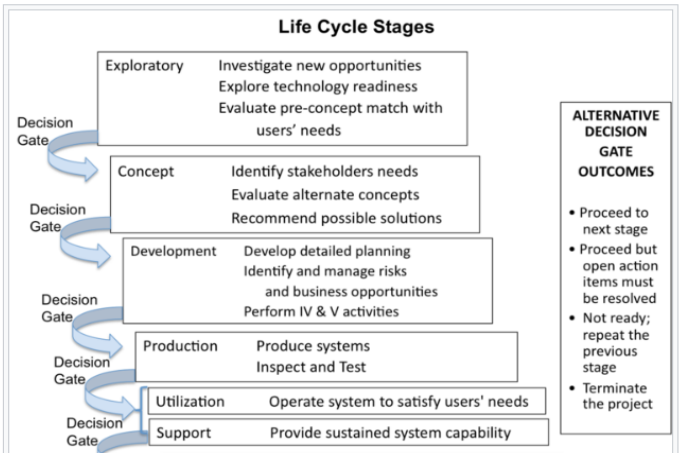
https://www.nytimes.com/2019/08/21/magazine/f35-joint-strike-fighter-program.html

https://www.defensenews.com/smr/defense-news-conference/2017/09/06/f-35-program-office-floats-new-agile-acquisition-strategy/

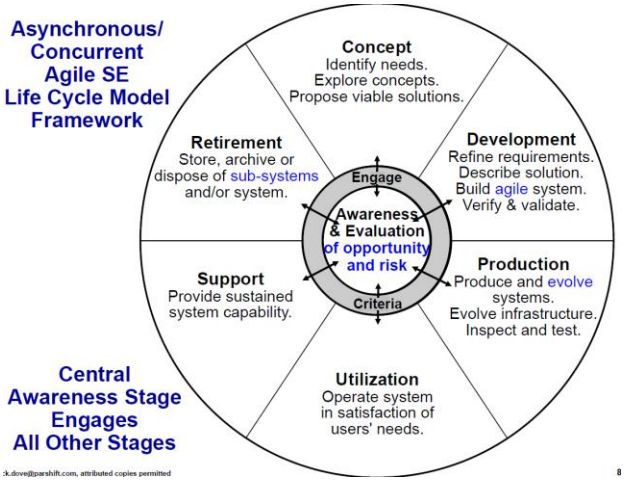
As JSF program leaders look to agile methods to improve engineering development, other systems engineering practitioners are taking notice. This article presents several examples of projects – systems engineering or otherwise – that have successfully implemented elements of agile software development in non-software domains.

An Agile Systems Engineering Framework

The life cycle stages of a traditional systems engineering project as depicted in the figure resemble a waterfall. Starting with this diagram, the *Agile Systems and Systems Engineering* working group within INCOSE has developed several tools for implementing a more agile approach to accomplishing systems engineering tasks. Employing these tools enable systems engineers to accomplish life cycle tasks with agility, and to produce products that exhibit agility in a changing environment.



https://www.sebokwiki.org/wiki/System_Life_Cycle_Process_Models:_Vee

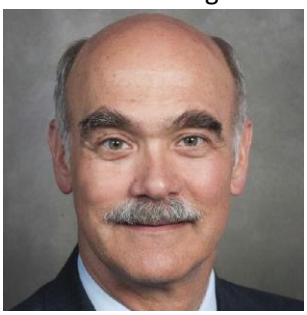


https://www.incose.org/incose-member-resources/working-groups/transformational/agile-systems-se

The working group, led by Rick Dove, has also revised the life cycle stage model to incorporate the importance of engagement and the focus on awareness within and between stages. The revised model is called the *Asynchronous / Concurrent Agile Systems Engineering Life Cycle Model Framework.*

Perhaps the addition of the *Engage* section in the new framework is the most crucial. The most important practices and ceremonies of agile development have a common factor, which is that they enable and encourage a high level of engagement. This engagement includes internal employee engagement within the development team, and external engagement between the development team and customer community. Successful projects require both.

When considering the adoption of agile development methodology, a fair question to ask is: Can Systems Engineering be agile? INCOSE Fellow Ron Carson asks just such a question in his 2014 INCOSE Symposium paper *Can Systems Engineering be Agile? Development Lifecycles for Systems, Hardware, and Software.* https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2334-5837.2013.tb03001.x



Dr. Ron Carson

Ultimately, his answer to this question falls somewhere between "no" and "it depends." In tackling this question, Carson also elaborates and appends to the decision criteria first declared in Boehm and Turner's book *Balancing Agility and Discipline: A Guide for the Perplexed*.

Suitability Factors	Agile	"Plan-driven" (Traditional SE)
Criticality	Low criticality	High criticality
Personnel	Senior developers	Junior developers
Dynamism	Requirements change often	Requirements do not change often
Size	Small number of developers	Large number of developers
Culture	Culture that thrives on chaos	Culture that demands order

https://en.wikipedia.org/wiki/Agile_software_development

When considering criticality, "low criticality" doesn't mean that agile projects are unimportant. Only that, as Carson explains, "Agile methods seek to minimize time to market," whereas "traditional systems engineering seeks to avoid breach of contract."

Likewise, SE doesn't require "junior developers," but the rigor and formality of systems engineering better tolerates less-seasoned developers.

Oft-changing requirements is the wheelhouse characteristic of agile development, and helps explain why the philosophy took hold most firmly in the software field. As software hopped residences from mainframe computers, to personal desktops, to laptops, to tablets, to our pocket video recorders/web portals, software survival requires adaptability to changing requirements, environments and culture.

The benefits of small developer teams will be discussed shortly, but first, the agile manifesto provides an informative introduction to the philosophy.

What's so Funny 'Bout Sprints, Scrums and Iterations

Seventeen thought leaders of the software development discipline co-signed The Agile Manifesto, which clocks in at just under seventy words (plus twelve appended principles) https://agilemanifesto.org/ *Scrum* is the most implemented strategy for agile software development, and defines three different manager roles.

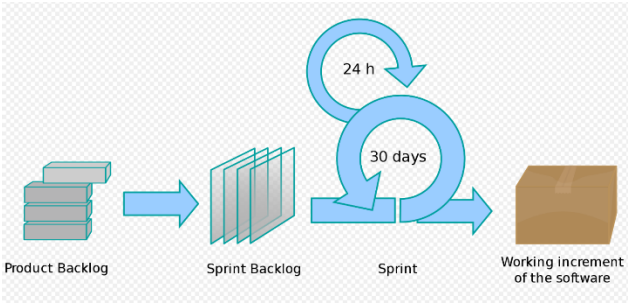
The first manager is called the Product Owner, and she is responsible for the management of the product. Her focus is acquiring the best product from the development team, as quickly as practical. The second manager is called the Scrum Master, and he is responsible for the management of the process.

Scrum masters schedule and facilitate the various meetings and ceremonies, remove roadblocks from the development team's progress, and make sure the processes are followed so that the team operates efficiently and effectively.

The third type of manager is the manager of the people. Contrary to belief, scrum is not missing a people manager. However, this role and its responsibilities are performed by the team itself. Decisions about whether and how to revise processes, handling conflicts, estimating work, assigning tasks; all are managed by the team members as a group. Why would a company dare trust its development team to manage itself?

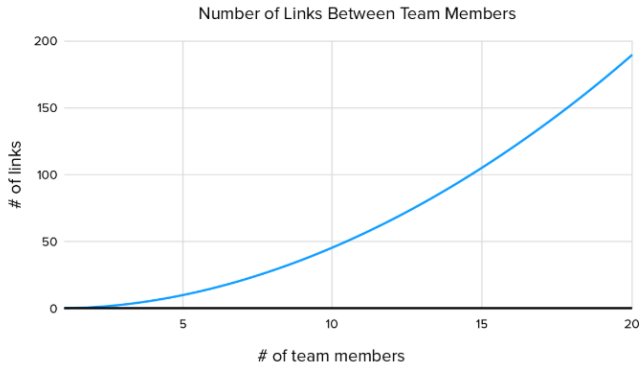
It turns out that employee engagement is highly correlated with an employee's sense of control over – and ability to manage – his own work. This also helps explain why agile development requires senior developers instead of junior developers. Senior developers are better able to perform tasks with no "chain of command" oversight. According to Gallup, engaged employees are more productive, and less likely to leave a company or team. Actively disengaged employees are more likely to be absent, quit, steal from the company, and degrade team morale. Especially in "work of the mind" vocations like engineering, employee engagement is *critical* to project success and company profitability. https://www.gallup.com/workplace/229424/employee-engagement.aspx

In the Wikipedia entry for scrum software development, the scrum lifecycle shows the primary artifacts and activities of scrum. Opinions vary about the optimal size of a scrum team, but all experts concur that the team size must be *small.* Nine people is a valid upper limit.



https://en.wikipedia.org/wiki/Scrum_(software_development)

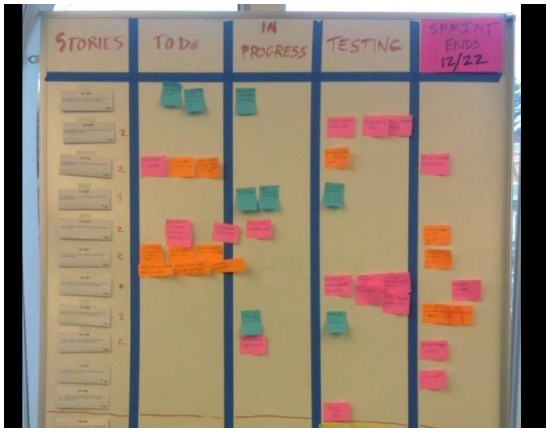
Why nine people? This ceiling is again related to employee engagement. Too many people providing status in the daily stand-up meeting make the meeting drag too long. Nine people is about the upper limit that can fit in a standard conference room. And as the chart demonstrates, the number of one-to-one relationships in a team do not increase linearly as the team size grows. The human mind can only navigate so many relationships before communication begins to suffer.



https://www.toptal.com/product-managers/agile/scrum-team-size

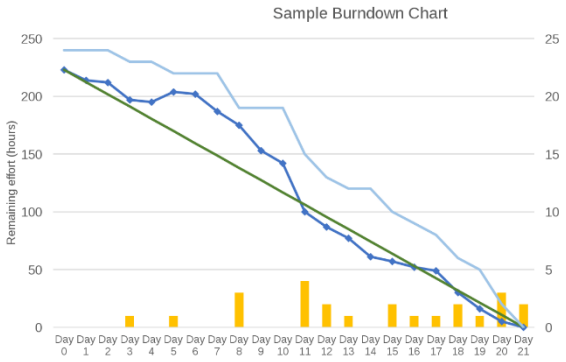
Team size also keeps the scrum task board small and manageable. With a small team and a short sprint duration of two weeks, each and every team member can wrap his mind around the scope of work for the entire team and the entire sprint. Rare is the developer who knows everyone on the program and what everyone is doing. But everyone on a scrum team can understand the work that everyone *on this team* does for the entire sprint. This is only possible with a small team.

Along with small teams, scrum works by requiring tasks to be small. When tasks are written and scoped in such a way that an uninterrupted developer can accomplish them in less than a day, the tasks flow across the task board from left to right like so many leaves blowing in the wind. Each time a developer moves a task a few inches into a new column, it sparks a tiny sense of accomplishment in the entire team. This rivulet of adrenaline and dopamine is enough to – you guessed it – keep employee engagement high. This degree of visual status and progress is impossible with a large, thousand-task and year-long Gantt chart.



A typical scrum task board

The preferred progress metric for a scrum team is the burndown chart. Scoped to a single small team and a single short sprint, burndown charts – prominently displayed for the entire team to notice – allow all team members to visually digest and follow everyone's progress, as a team. This too optimizes employee engagement, since every employee's daily accomplishments show prominently in the shared movement of the team's progress metric.



It's not just employees that maintain engagement using scrum. As seen in the scrum lifecycle, the first artifact is the product backlog. This is a list of small tasks, maintained by the product owner, and prioritized by the product owner and customer community in order of business value. This prioritization of the backlog ensures that the customer receives her most important functionality first, and quickly. The benefit of the prioritized backlog for customer engagement is that it puts the customer in a good mood, right away. Since scrum relies on continuous customer feedback and involvement, the advantage of a customer in a good mood cannot be overstated.

Empirical Examples

Many examples from systems engineering literature demonstrate how agile-inspired practices succeed in domains outside of software. This section lists several examples, ordered by SE lifecycle stage. The first example, though, is not from the engineering field at all. The Exploratory lifecycle stage example comes from the world of entertainment, specifically public radio.


Eric Nuzum

Eric Nuzum is the former vice president of programming at National Public Radio. Since NPR provides digital content in the form of podcasts and radio show streaming, the organization employs a small software development team, which uses agile development to plan its work. Nuzum was impressed by the software team's productivity, and decided to adopt some of its agile practices for the task of new radio show production.

The standard way of creating a new NPR radio show is for a creative team or person to conceive an idea for a radio show, then pitch the idea to an executive. If greenlit, the show gets a budget, talent is hired, the show gets produced, promoted, and then rolled out to many NPR stations. The process is expensive and risky.

Nuzum decided to incorporate the agile philosophy of deploying to production quickly and inexpensively, then iterate often. So he greenlit many small and inexpensive shows, usually either live-audience shows or shows based on other media. The shows were produced for small runs of six to ten episodes, and rolled out to only a few stations. The shows would then solicit feedback from listeners and station managers, and tweak the shows based on that feedback.

https://www.poynter.org/reporting-editing/2012/how-npr-benefits-from-agile-project-development-you-can-too/

https://www.computerworld.com/article/2505876/npr-adopts-agile-like-method-for-program-development.html

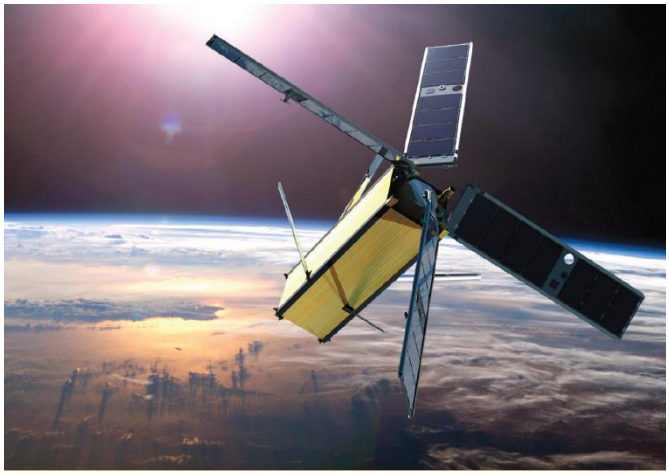
https://www.niemanlab.org/2012/04/agile-social-cheap-the-new-way-npr-is-trying-to-make-radio/

NPR created several successful radio shows using this method. Among them are the podcast *How To Do Everything* https://howtodoeverything.org/, the *TED Radio Hour* https://www.npr.org/programs/ted-radio-hour/ based on popular TED talks, and the comedy/trivia show *Ask Me Another* https://www.npr.org/programs/ask-me-another.

The Midwest Gateway Chapter's own Rob Simons shares his experiences with agile implementation for both systems engineering teams and projects at Boeing. His list of hard-earned lessons, from the Concept and Development lifecycle stages, is below:

1. An effective SM (Scrum Master) is crucial
2. Start Scrums mid-week and everyone stands during the Daily Scrum (huge timesaver)
3. PO availability (or being reachable) (popping in irregularly slows the team)
4. The SM should NOT be on the Dev Team and no sharing of SM or DevTeam members across concurrent activities
5. Tools (i.e., JIRA) good for sprint statistics, but better to discuss and/or show periodically at daily scrum (ex., burndown)
6. Managed our Scrum using backlog scope and scoring to accomplish sprint execution (i.e., story structure to meet scope)
7. Since SE doesn't release 'working code' like SWE, demo artifacts were 'stand-alone' artifacts (e.g., analyses, trade study)
8. At Sprint Demo (i.e., show/demo/explain the artifacts) everyone must focus on staying within the time-box, otherwise the demo takes all day --- in other words, scope to the sprint activity and not the larger system)
9. Don't let Sprint Retrospectives become just summaries --- try to capture what works and what doesn't

Another Development stage example comes from the Johns Hopkins University Applied Physics Lab. http://www.parshift.com/s/140630IS14-AgileSystemsEngineering-Part2.pdf



https://www.baltimoresun.com/health/bs-hs-apl-cube-satellites-20140110-story.html

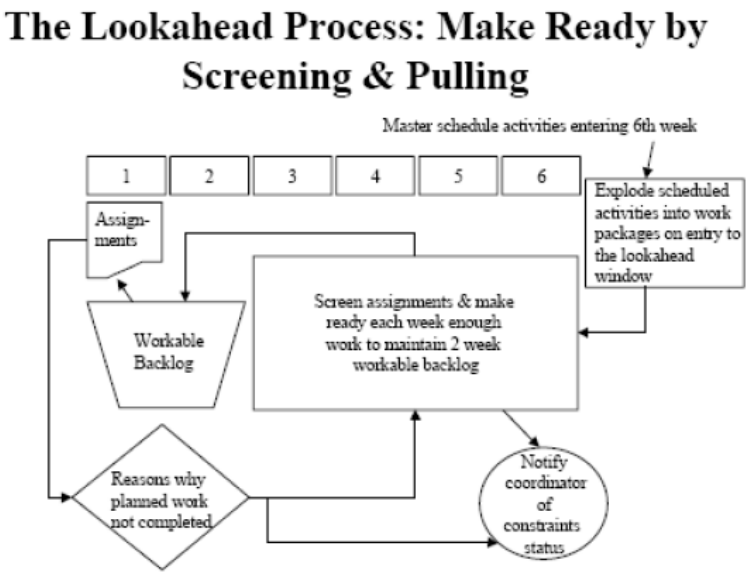
In this organization, "cubesats," or tiny satellites (10cm X 10cm X 30cm) are developed by the development teams in the Multi-Mission Bus Demonstration program. These cubesats provide myriad services for customers in the university, government, and military communities. But the satellites piggy-back onto the payload of already scheduled rocket launches, so the development team cannot be picky about payload attachments or container configurations. This environment necessitated an agile mindset during development.

In order to meet this required agility, the MBD program implemented several agile development strategies. The program manager, called the "Sherriff," performed the duties of a product owner. Each of six engineering teams was led by a team lead, or "Deputy." The engineering teams consisted of Payload, Electrical, Software, Mechanical, Ground and Navigation Control, and Avionics, and were kept to a small number of engineers. These teams called themselves "Posses."

Daily stand-up meetings, or "Round Ups," focused on daily priorities and issues. One scrum task board displayed progress for all six development teams, and the teams kept sprints to *one day* duration.

A Production stage example comes from the Lean Construction Institute, within the Civil and Environmental Engineering department of the University of California, Berkeley. There, Professor Glenn Ballard teaches the benefits of the Last Planner system.

In the construction of a building like a skyscraper, an overall schedule can include several years of tasks. But the hard works comes in planning the current week's tasks. Ballard calls this scheduler the Last Planner. Her job is ensuring that all crews are working, all materials are present, and all predecessor tasks are complete.



The Last Planner system

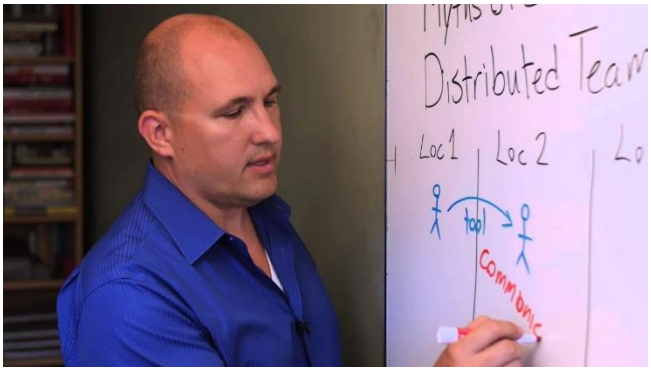
This is accomplished by limiting the scope of tasks to just those planned for up to six weeks in the future. These tasks are exploded into small work packages and enter a task

backlog called the “lookahead window.” The lookahed window holds tasks for up to four weeks, where they are monitored to ensure applicable stakeholders remove any constraints and resolve any dependencies.

Workable tasks are then put into another backlog, called the “workable backlog.” The workable backlog keeps at least 2 weeks of work packages at all times.

The Last Planner then populates the current week’s assignments, drawing from the workable backlog. She also recycles incomplete assignments back into the lookahead window for issue resolution, and communicates master schedule changes to supervisors.

This process mirrors many scrum practices, such as maintaining prioritized backlogs, sprint planning, keeping tasks small, and scrum master process ownership.



Mishkin Berteig

An example of agile methodology in the Retirement stage comes from agile consultant and former Charles Schwab Chief Architect Mishkin Berteig.

In the referenced 2015 article, Berteig describes a project in which a company migrated its data warehouse from Oracle to Teradata. It was essentially a retirement stage effort performed in parallel with a deployment effort.

In order to track the work, Berteig advised using a product backlog with 25,000 data elements comprised of tables, views, and scripts. The product owner and Berteig devised a calculation to prioritize the data elements based on business value, using processor speed and disk space as calculation inputs.

http://www.agileadvice.com/2015/03/14/scrumxplean/scrum-data-warehouse-project/

Another backlog tracked the five Oracle licenses to retire, one at a time, throughout the migration effort. The project employed the agile concepts of prioritized product backlogs, burndown charts, and small development teams.

Putting It All Together

In summary, the advantages of software development can be realized in disciplines outside of software – including systems engineering – as long as the underlying philosophy is courted. The key practices all involve *engagement*, of both the development team and the customer community. Key practices include:

- Keep the teams small. A small number of developers on a team provide a variety of employee engagement benefits, which allow the team members to keep communication effective, work comprehensible, and progress perceptible.
- Keep the tasks small. Keeping tasks focused, short, and simple allows for common understanding, and allows tasks to move briskly across the task board. This aids in understanding and motivation.

- Get out of the way. Allowing development teams to managing themselves, and their own work, keeps employees motivated, engaged, productive, and happy.

The success of agile development in many systems engineering projects shows that – for some projects meeting certain criteria – agility is not a fluke. Whether building a fighter aircraft or a comedy radio show, agile development is a tool all systems engineers should keep handy in their toolboxes.