Appendix A: Project Outcomes of Agile vs. Waterfall Methods for Software Development

Agile methods are more successful than traditional Waterfall methods for software development projects of all sizes (Standish Group, Reference 1). This report looked at over 50,000 software projects from 2011-2015, segmented by agile vs. waterfall project method. In each size category (small, medium, and large), projects employing agile methods succeeded in meeting project goals more often than waterfall projects, and completely failed less often (defined as cancelled or never adopted by users).

Waterfall projects employ sequential phases for requirements analysis, design, coding, and testing, with each phase having to be completed before the next one starts. Value is delivered to the customer only at the end of the whole sequence. A corollary practice is attempting to complete detailed design before beginning coding or construction (referred to as Big Design Up Front, or BDUF).

In contrast, agile methods (which include Scrum, XP, DevOps, others) employ iterative, incremental development and delivery of features. Value is delivered to the customer as early as at the end of each iteration (often 1 to 4 weeks long). Frequent feedback from customer use of the product drives adaptation of the delivered product design to better meet actual “real-world” customer needs.

As initially laid out by the founders of the agile movement and the Agile Alliance, core agile principles include (from https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/):

1. “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”

These principles have now been applied to more than just software development. Agile methods are not a silver bullet for every project. Traditional methods work well when the work is repeatable and when the end-product’s design is already well-known and fixed. However, in projects where the deliverable is new or complex, and learning more about the exact optimal design during the project is likely, agile methods consistently outperform a traditional development approach, in terms of time-to-delivery, cost, and quality (fitness-for-use) of the delivered product.
Appendix B: Model-Driven Development

Diagram and explanatory text adapted from: Kannan V, Fish JS, Mutz JM, et al. Rapid development of specialty population registries and quality measures from electronic health record data: an agile framework. Methods Inf Med 2017;56(Open):e74-e83 doi: http://doi.org/10.3414/ME16-02-0031 (Reference 26, online Supplemental Material),

The above is an adaptation of the process outlined by Rosenberg in Use Case Driven Object Modeling with UML: Theory and Practice (Ref 37). Single clinical decision support (CDS) tools may employ only selected parts of this. However, the process can scale to handle large initiatives containing multiple types of features, such as a combination of clinical decision support, ordering, documentation, and reporting features.

Project requirements can be captured initially as a high-level User Story with accompanying Acceptance Criteria. Within an iteration, analysis and design may include a Domain (UML Class) Model to classify the structural aspects (“nouns”) of the project, and their relationships. A UML Use Case Model unambiguously catalogs the dynamic behavioral aspects (the “verbs”) of the CDS and other features being developed. This model also depicts the user role(s) deriving value from each Use Case. A User Interface Model (e.g. of a clinical decision support alert) can be created early as a wireframe mockup. For each use case, a Use Case Text can be written,
often in a “user does this, system does that” format, elaborating both the basic path for the use case and any alternate paths. This format leads naturally to test scripts, and—when elaborated with system screenshots—to a “tipsheet” for instructional purposes on how users will interact with the system.

Additional optional models are employed where helpful. UML Activity (swimlane workflow) diagrams help depict the sequence of activities in a defined workflow, including the role responsible for each step. A swimlane diagram can provide valuable context for the envisioned location of a given CDS feature within a clinical workflow. Decision trees (or tables) unambiguously describe the logic to be followed by a rule, and frequently aid in modeling desired CDS behavior.

To accomplish modeling within short iteration timeframes (see Figure 2), Ambler’s “agile modeling” principles and practices (Ref 29) prove valuable, including “using the simplest modeling tools possible” (often whiteboard drawings), “creating simple content”, “modeling with others”, “active stakeholder participation”, and “creating several models in parallel”. Some high-level overall architectural modeling can be performed initially. However, large user stories are split into small enough deliverable “chunks” (each with their own user story), such that detailed analysis and design modeling, build, and testing can be performed within a single iteration, and the working feature demonstrated at end-iteration.

For additional information on model-driven development of clinical decision support, see also: Kannan V, Willett DL. Agile clinical decision support development. Agile Alliance Experience Reports 2016; 28 March 2016. https://www.agilealliance.org/resources/experience-reports/agile-clinical-decision-support-developments/ (Reference 23).
Appendix C: Scrum

Scrum Process Model. Adapted From: Essential Scrum: A practical guide to the most popular agile process. Kenneth S. Rubin, Addison-Wesley 2013 (Reference 31).

In the Scrum agile process, during a time-boxed iteration (often 1-4 weeks), inputs (features) for development during that iteration are transformed into completed, tested, working output(s) demonstrated at end-iteration. Feedback from the end-iteration demo and retrospective (on what worked well and what didn’t) leads to iterative adaptation of both the Product being produced and the Process being used by the scrum team. Thus, no two Scrum teams are likely to follow the exact same process, as they adapt the way they work to their specific situation. Still, agile methods provide guiding principles (including the Agile Manifesto https://www.agilealliance.org/agile101/the-agile-manifesto/) and the Agile Alliance principles listed in Appendix A. The Scrum methodology includes a set of specific practices helpful in adoption: see for instance https://www.scrum.org/resources/what-is-scrum.

Iterations in Scrum are called “sprints”, originally one month long when Scrum was first developed, though shorter time frames of 1-2 weeks are now common. One core concept is that of the “product backlog”, which includes all possible future work, and can be expressed in the form of User Stories. Some potential user stories near the bottom of the backlog and not yet ready to be scheduled may be of large size (in Story Points, for instance). User stories nearer the top of the backlog and being prioritized for coming iterations will be split into smaller stories each able to fit into one iteration. The Product Owner can re-prioritize the Product Backlog any time.

The set of stories to be delivered in a given iteration is the Sprint Backlog. The Sprint Backlog is solidified during a Sprint Planning meeting, where the Scrum Team can evaluate the size of the highest priority items in the Product Backlog, and determine how many items they are able to commit to delivering. Ideally this is guided by recent historical data on the number of user story points completed per iteration, or team “velocity.” Once a
given sprint is underway, the current Sprint Backlog generally cannot be added to (and if so, that is called “breaking the sprint,” which is a major event).

During the sprint, a “daily scrum” is held, guided by a Scrum Master, and meant to be no more than 15 minutes long. At the scrum, each Scrum Team member shares briefly what tasks they’ve accomplished since the prior day, what they commit to completing by the next day’s scrum, and any roadblocks they’re running into for which they may need help in overcoming. Progress during the sprint may be measured by a “sprint burndown chart” showing daily progress in the total number of hours of work remaining in the sprint. This is based on hour estimates for all remaining tasks, as judged by team members. Credit on the burndown chart is given only for completed tasks. At the sprint’s end, the team demonstrates working features (user stories) developed during the iteration to stakeholders at a Sprint Review meeting. The Product Owner (accountable for maximizing value from the product) and team incorporate feedback on the features demonstrated for reprioritizing and re-shaping the Product Backlog as needed. The Scrum Team also holds a Sprint Retrospective to reflect on what worked well and what did not, to brainstorm how to adaptively improve their process for the upcoming sprints.

For additional information, see also:

- Reference 31: Essential Scrum: A practical guide to the most popular agile process, by Kenneth Rubin (500 pages)
- The Elements of Scrum, by Chris Sims and Hillary L. Johnson (184 pages)
- Scrum: a Breathtakingly Brief and Agile Introduction, also by Chris Sims and Hillary L. Johnson (54 pages).
### Appendix D: SPIDR acronym for ways (dimensions) to split stories to fit into iterations

| Dimension | Description | CDS User Story-Splitting Example |
|-----------|-------------|----------------------------------|
| **Spikes** | Spikes are small, prototypical implementations of a functionality that is typically used for the evaluation and feasibility of new technologies. | When creating your first EHR-based registry, in one user story construct a simple rule that will populate patients on the registry based on the diagnosis of atrial fibrillation on the Problem List. This proves the ability to successfully include patients and display them on a registry list report. In subsequent user stories, refine the inclusion criteria to more completely capture patients to display on rows, and add rule-based columns to display if the patient is currently on an anticoagulant, or has certain co-morbid conditions. |
| **Paths** | If there are several possible alternative paths in a user story, one option is to create separate user stories from some of these paths. | To support risk-appropriate VTE prophylaxis on inpatients, a set of questions making up the VTE risk-scoring system needs to be answered, if the data is not already in the EHR. One option is for the patient to receive a patient-portal questionnaire prior to a scheduled admission. Another path is for pre-operative clinic assessment staff to complete the patient while evaluating the patient. A third path is for the admitting physician to supply any remaining missing information during the admission process. Each of these paths could be implemented as a separate user story, even if the data is being written to an identical storage place in the medical record for any given scoring question. |
| **Interfaces** | Split by type of interface device, or by platform (e.g. iOS vs. Android) to support. Or split by features available on user interface (UI). | For a CDS advisory for chronic opioid management, initially offer option on the advisory UI to place an Order Set with appropriate medication options, counselling referrals, and documentation. In a later story, add to the UI a display of recent prescriptions with morphine equivalent doses. OR For a patient reminder system serving as patient clinical decision support, deliver it first by the EHR-based patient portal. Later add support for a locally-branded mobile app connecting to the EHR via FHIR, and/or for text message notifications. |
| **Data** | Split by a sub-range of relevant data. | Suggest addition of missing Diabetes Mellitus on Problem List initially based on Past Medical Hx entry only. In a later user story, also suggest addition of DM diagnosis based on Hgb A1C value, and/or by presence of long-acting insulin on medication list. |
| **Rules** | Split by business rule. Deliver a general feature initially to provide value to users early, then refine by adding tighter business rules. | During a relatively sudden national shortage of IV pain medication, when ordering a dose on an inpatient, offer substitution of an oral pain medication instead to preserve limited local stock. In a later user story, exempt patients in the PACU, since oral pain medications are rarely appropriate in that setting. OR When recommending a NOAC for anticoagulation of atrial fibrillation but which is contraindicated in patients with a certain level of renal dysfunction, initially have the order set display for all patients, with a comment to avoid ordering in patients with moderate or severe chronic kidney disease (CKD). In a later user story, implement a business rule that doesn’t display this recommendation when the patient’s lab values and/or problem list indicate they have CKD in the contra-indicated range. |

Adapted from: [https://blogs.itemis.com/en/spidr-five-simple-techniques-for-a-perfectly-split-user-story](https://blogs.itemis.com/en/spidr-five-simple-techniques-for-a-perfectly-split-user-story)
Appendix E: Use Case Diagram for Rheumatology Registry Project

For large specialty registry development projects, we often drew two use case diagrams, one for data collection related use cases, and one for reporting use cases (see References [26,49]). Shown above is the data collection use case diagram for the Rheumatoid Arthritis Registry project reported on in this paper.

One of these use cases represents clinical decision support (CDS): “Alert that DMARD (Disease Modifying Anti-Rheumatic Drug) not on the Med List”. The use case diagram shows this alert will be delivered to the rheumatologist in clinic and not to nurses or medical office assistants (MOAs). Accordingly, the [type of user] field in the User Story for this CDS alert is written:

As a rheumatologist,
I want to be advised if my patient with rheumatoid arthritis is not on a Disease Modifying Anti-Rheumatic Drug (DMARD),
so that they may receive optimal therapy and can experience symptom improvement.

Addition of Acceptance Criteria further specified additional success criteria for the alert, such as under what conditions it should display, options to be available on the user interface, etc.
Appendix F: Additional Models for VTE Risk Project

1. Swimlane Workflow Diagram

The Venous Thromboembolism (VTE) Risk and Prophylaxis Redesign initiative was a large project with multiple features including CDS to be developed, depicted on a Use Case Diagram in Figure 4 (as well as on a Feature Breakdown Structure in Figure 5). Those models helped to visualize the project scope and create a shared vocabulary for features to be prioritized and scheduled into iterations for development. However, additional modeling proved helpful to unambiguously clarify other aspects during project analysis and design.

Use Case Diagrams can depict event-driven activities that don’t always follow a defined sequence. But when depicting a defined process, a Swimlane Workflow Diagram shows the desired sequence of activities along with the role responsible for each.

For the above scenario of a Pre-op Clinic Visit leading to Inpatient Surgery, the swimlane workflow diagram shows the planned sequence of activities—including where in this workflow a CDS advisory (BPA) should prompt the Physician or APP to Review and Update the VTE Risk Form, if not yet completed.
2. Decision Tree for a Best Practice Advisory to prompt ordering of VTE prophylaxis on admission

For modeling clinical decision support tools that employ deterministic logic, a decision tree shows under exactly which conditions the CDS should appear versus not appear to meet the user story’s acceptance criteria. (Decision tables also show equivalent information.) During rapid-cycle design and development of CDS tools such as best practice advisories (BPAs), decision trees can aid in co-designing with clinicians the precise logic under which the BPA should display.

To then depict how the user can interact with the CDS tool after it has appeared requires different model types. In one use-case driven design approach (Reference [37], a mockup of the proposed CDS tool’s user interface (UI) is drawn, then use case text is written in the form of “user does this, system does that” for the main path and for alternate paths the user might take when interacting with the displayed UI. (See also Appendix B).
3. Object Diagram for the same Best Practice Advisory

The decision tree logic and other models are useful input when creating a diagram of the objects needing to be constructed or configured within the EHR. Compared with a simple list of relevant EHR records (a “build tracker”), an object diagram for a CDS tool offers some advantages:

- can be drawn before the objects are created, enabling peer review of design
- depicts the flow of dependencies between objects, which can be useful in guiding construction, and later in understanding and/or modifying the design
- makes more apparent any opportunities for re-use of existing objects
- helps in recognizing design patterns potentially useful for future CDS tools.

Ultimately, the objects collectively need to produce system behavior that fulfils all the requirements of the Acceptance Criteria and any acceptance tests, after which the User Story can be marked as complete.