Agile methods in biomedical software development: a multi-site experience report

David W Kane, Moses M Hohman, Ethan G Cerami, Michael W McCormick, Karl F Kuhlmman and Jeff A Byrd

BMC Bioinformatics 2006, 7:273

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Emergence and Iteration

- A fundamental change in programming philosophy: treating it like an exploratory process (i.e. a science) rather than the traditional "figure-it-all-out-first" approach.

  Much like object-oriented programming overtook traditional procedural programming by managing to the pieces of the system, agile development pays attention to the details and exploration made available by iteration, instead of following a "monolithic," one-answer-fits-all approach.

Extreme Programming

- Kent Beck's Extreme Programming (XP) is one of the most well known, and also one of the most controversial of the agile methods. Beck took a number of well-known practices and "turned them up to 10" [13]. For example, in XP, peer reviews are implemented as pair programming, i.e. continuous peer review. The method assumes that software is being developed in a dynamic environment, so the software development approach needs to be able to adapt to this rapid change.
  XP is organized around short iterations, typically one or two weeks long. Features in XP are described as user stories. Programmers estimate the effort to complete each story. Each iteration the customer gets an effort budget and selects a list of stories for implementation from the list of possible user stories. The customer bases the budget on the completed story velocity from the previous iteration. This means that, iteration by iteration, the customer directs the development team to work on the features most important and immediate to that customer.
  The engineering practices of XP are notable as well. The method forgoes a distinct design phase in favor of an evolutionary design approach for the software. The approach focuses on informal communication rather than formal documentation. XP makes extensive use of automated unit and acceptance tests. Programmers run these executable tests throughout the development process. When developing new features, the development team uses "the simplest [design] that could possibly work" [13]. When adding a new feature that requires extension of the existing design, programmers refactor the code first to improve the design, and then add the new feature. Refactoring improves the internal design without changing the external behavior. The automated tests provide verification that the behavior of the code has not changed after refactoring.
  Developers using XP continually shift back and forth between refactoring activities and new feature development activities. XP uses the combination of testing and refactoring to make the software malleable to accommodate rapid requirements change.

Scrum

- Scrum is an agile method that focuses on project management [28]. The key practices focus on management of feature backlogs. The mechanics of backlog management are similar to those found in XP, i.e. the customer regularly prioritizes features, and in each iteration the development team implements the top N features from the list. Iterations, called sprints, are 30 days long. In addition to selecting features for each sprint, the development team organizes several sprints into releases. Project managers use "burn down" charts to track progress within a sprint or a release. These charts provide a view of progress by plotting unimplemented features against time remaining in the sprint or release.

  Another key practice in Scrum is the scrum meeting. A scrum meeting is a daily stand-up meeting for the development team. In the meeting, each member of the team reports what they did since the last scrum meeting, what work is planned before the next scrum meeting, and obstacles. The meeting is short and facilitates open and frequent communication in the team.

  Scrum does not specify a particular engineering approach; the approach says nothing about testing or configuration management practices. Some groups apply XP-style engineering practices to complement their scrum practices, but other variants are common as well.

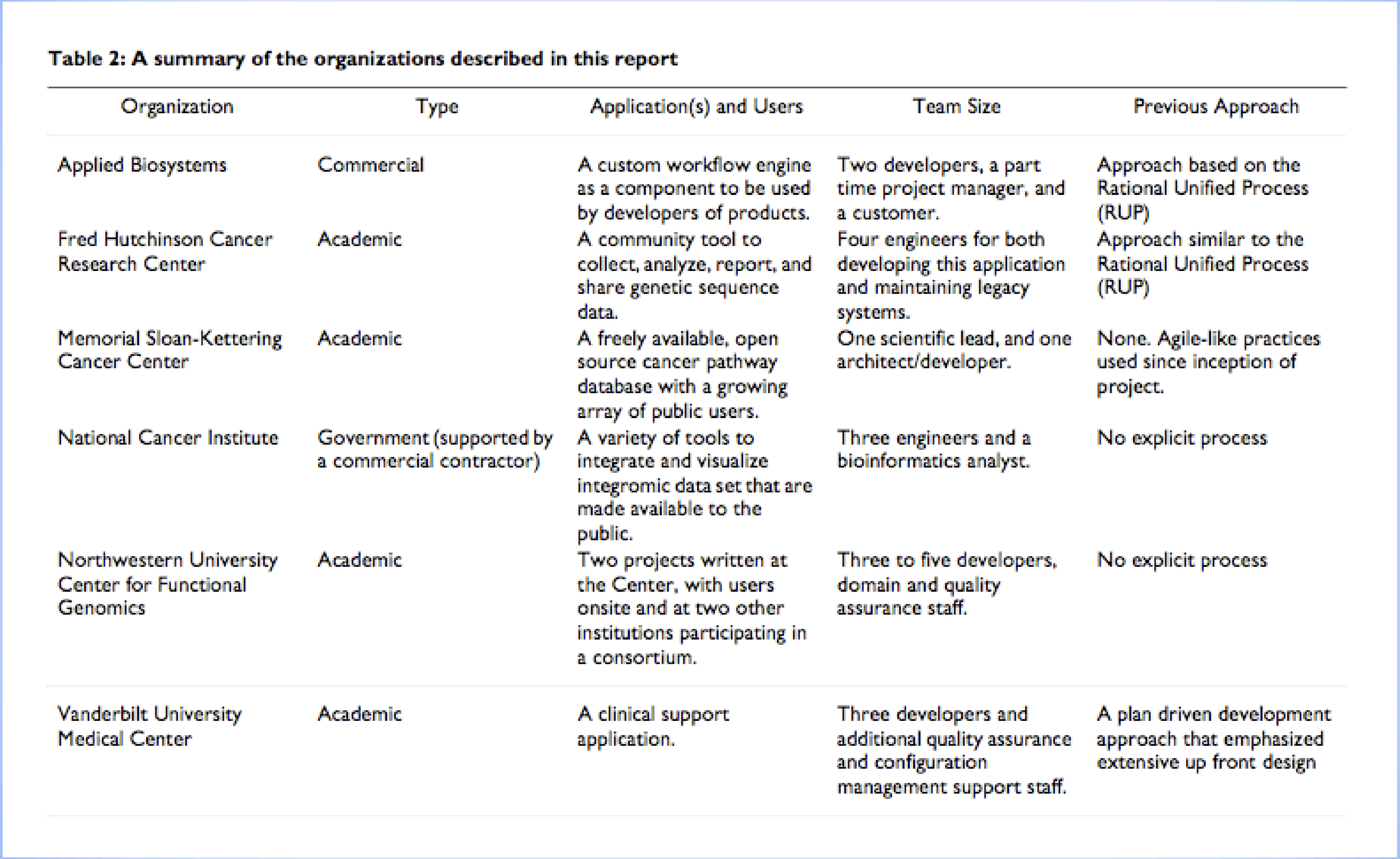

Table 2: A summary of the organizations described in this report

Organization	Type	Application(s) and Users	Team Size	Previous Approach
Applied Biosystems	Commercial	A custom workflow engine as a component to be used by developers of products.	Two developers, a part time project manager, and a customer.	Approach based on the Rational Unified Process (RUP)
Fred Hutchinson Cancer Research Center	Academic	A community tool to collect, analyze, report, and share genetic sequence data.	Four engineers for both developing this application and maintaining legacy systems.	Approach similar to the Rational Unified Process (RUP)
Memorial Sloan-Kettering Cancer Center	Academic	A freely available, open source cancer pathway database with a growing array of public users.	One scientific lead, and one architect/developer.	None. Agile-like practices used since inception of project.
National Cancer Institute	Government (supported by a commercial contractor)	A variety of tools to integrate and visualize integromic data set that are made available to the public.	Three engineers and a bioinformatics analyst.	No explicit process
Northwestern University Center for Functional Genomics	Academic	Two projects written at the Center, with users onsite and at two other institutions participating in a consortium.	Three to five developers, domain and quality assurance staff.	No explicit process
Vanderbilt University Medical Center	Academic	A clinical support application.	Three developers and additional quality assurance and configuration management support staff.	A plan driven development approach that emphasized extensive up front design

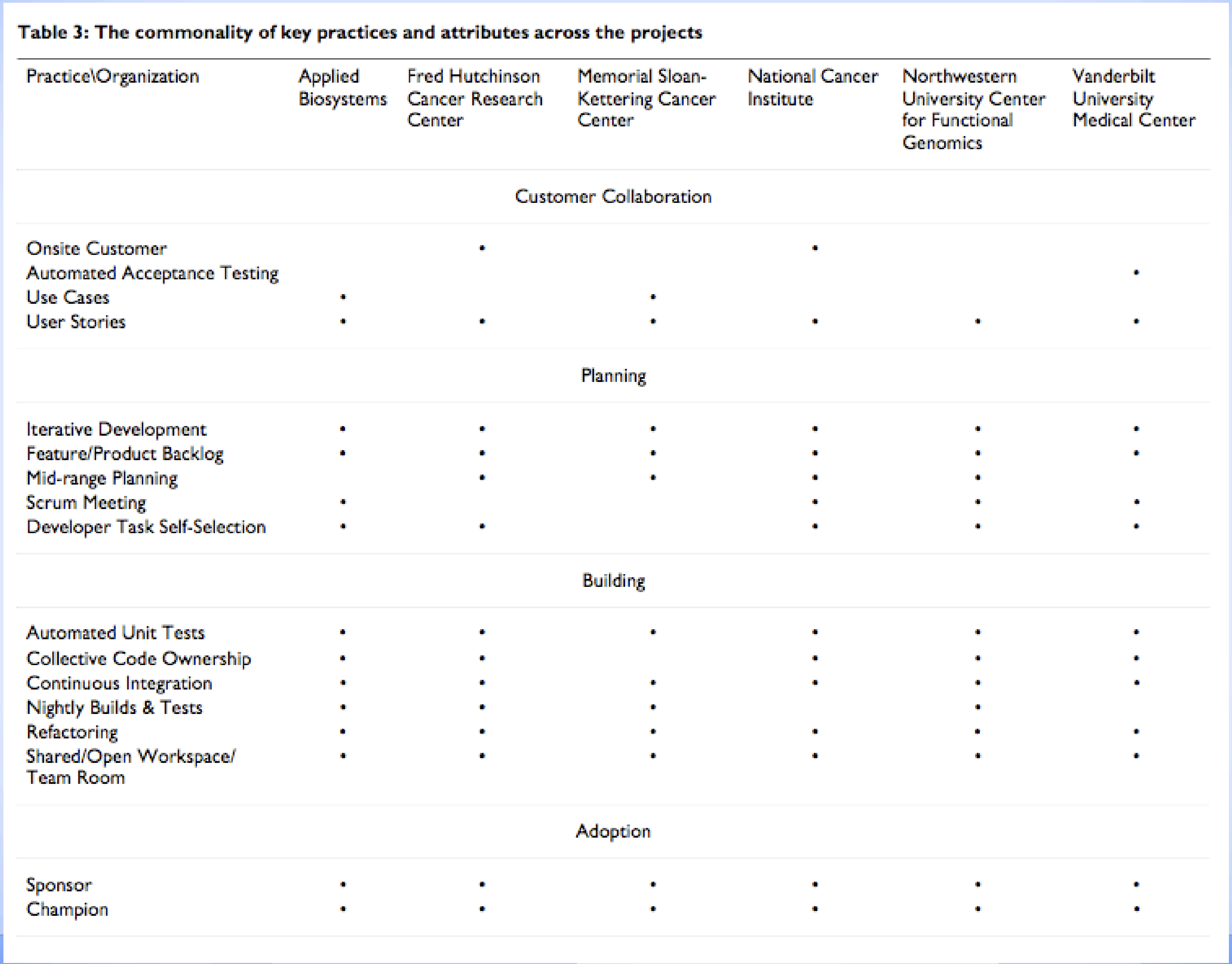

Table 3: The commonality of key practices and attributes across the projects

Practice\Organization	Applied Biosystems	Fred Hutchinson Cancer Research Center	Memorial Sloan-Kettering Cancer Center	National Cancer Institute	Northwestern University Center for Functional Genomics	Vanderbilt University Medical Center
Customer Collaboration						
Onsite Customer		•		•		
Automated Acceptance Testing						•
Use Cases	•		•			
User Stories	•	•	•	•	•	•
Planning						
Iterative Development	•	•	•	•	•	•
Feature/Product Backlog	•	•	•	•	•	•
Mid-range Planning		•	•	•	•	
Scrum Meeting	•			•	•	•
Developer Task Self-Selection	•	•		•	•	•
Building						
Automated Unit Tests	•	•	•	•	•	•
Collective Code Ownership	•	•		•	•	•
Continuous Integration	•	•	•	•	•	•
Nightly Builds & Tests	•	•	•		•	
Refactoring	•	•	•	•	•	•
Shared/Open Workspace/ Team Room	•	•	•	•	•	•
Adoption						
Sponsor	•	•	•	•	•	•
Champion	•	•	•	•	•	•

Survey & Results

- All teams fairly small (1-6)
- All used Java
- Software complexity arises from inherent complexity.
- More than one project at a time (2/3)
- No dedicated QA
- No safety-critical applications.
- Need for a close working relationship between biologists and developers. *Information asymmetry (e.g. "database")*
- Weekly meetings with stakeholders, plus collocation, periodic interviews, etc.
- Collaborative decision-making (good idea?)
- Independence and investment...

Agile Development Practices

- Capturing requirements (varying degrees, most based on customer feedback, use of web tools)
- Automated acceptance testing (FitNesse tool; demonstrates complexity of issue and value of customer interaction)
- Open workspaces (always developers, sometimes customers)
- Project planning and prioritization (ability to change fundamental assumptions)

Agile Iterations

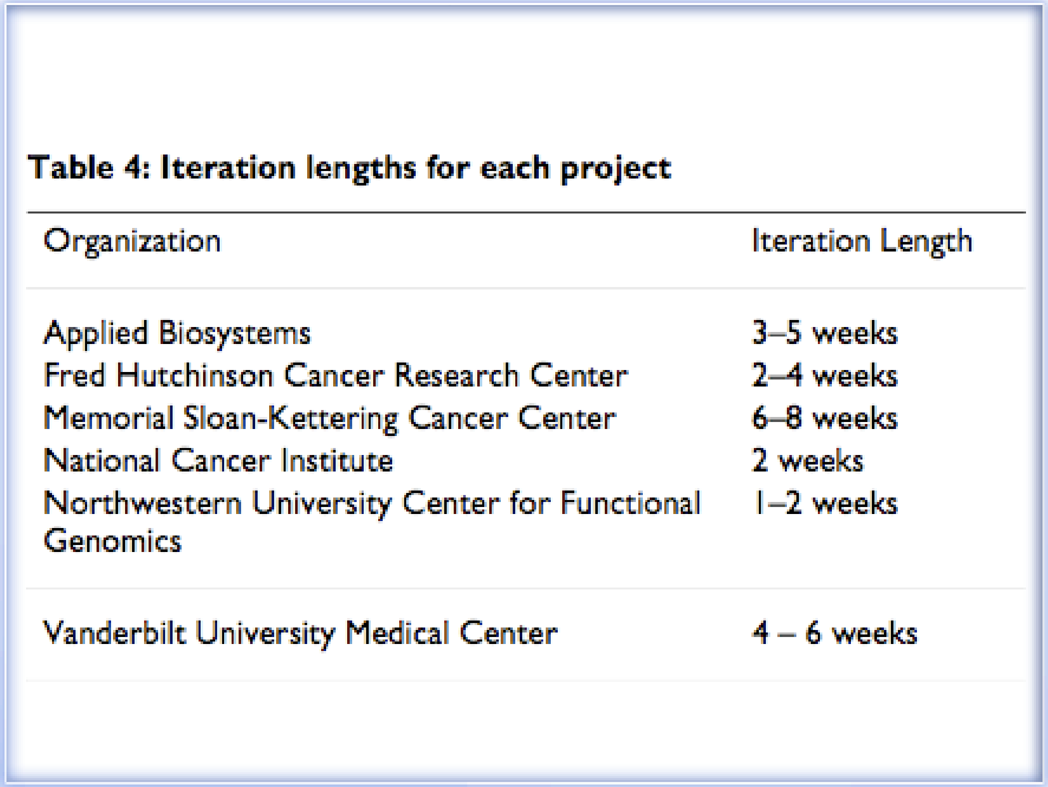

Table 4: Iteration lengths for each project

Organization	Iteration Length
Applied Biosystems	3–5 weeks
Fred Hutchinson Cancer Research Center	2–4 weeks
Memorial Sloan-Kettering Cancer Center	6–8 weeks
National Cancer Institute	2 weeks
Northwestern University Center for Functional Genomics	1–2 weeks
Vanderbilt University Medical Center	4 – 6 weeks

Agile Estimation

- Use of a feature backlog (plot features vs time)
- Velocity (Acceleration/Deceleration?)

Agile Release Planning

- Less value placed on fixed release dates
- More value placed on updates, bug fixes
- Limited staffing resources, multiple projects, but still releases were largely delivered on time

Agile Decisions

- Central authority vs autonomy and collaboration (consortia?)
- Coding practices (consistent across all groups, higher discipline/quality)
- Automated testing
- Refactoring (simplification and/or speed without the customers seeing it)
- **Pair programming**

Agile Adoption

- Champions and Sponsors!

Discussion

- Sea-change for development practices (interconnectivity?)
- Agile methods valuable for bioinformatics projects (?)
  - Adapting to change, enabling collaboration, need for software quality
- Common core practices: automated unit tests, continuous integration, feature backlog, refactoring, open workspace
- Augmented practices development (internal/external, improved prioritization tools, distribution)

Questions

- How does the agile process improve software development? Particularly for bioinformatics?
- What does agile sacrifice? Is this okay?
- Does agile benefit enough from infrastructure (e.g. rapid web deployment) to accept it as a distributed development method? Shaky?
- What about these unadopted practices (e.g., pairing)? Might they work if enforced here?
- Can we come up with ways to enhance agile even further? Where does agile take us next?
- What about agile for safety-critical applications? Still okay?

Appendix

The following is the list of survey questions asked of each author. Each interview took approximately one hour. The interviewer presented the questions over the phone (except when he interviewed himself), so they are not recorded in sufficient detail to stand by themselves as a written interview. Please email the interviewer (moses@moseshohman.com) with any questions.

For any question, if the answer has changed over time, please describe how and why.

1. Describe your project.
2. Describe your users, customers, whatever you call them. If your software is intended for a public audience, describe your user/customer representatives.
3. Did you join the project at the beginning or was the project already in development before you?
4. How long have you been with your project? How long has it lasted overall?
5. When did you introduce agile methods?
6. What existed before the introduction of agile?
7. Were you the main proponent of agile, or were there others? If others, describe.
8. How did you learn about agile?
9. Why did you introduce agile, and in what ways did you introduce it? What problems did you hope agile principles would solve? Did you solve these problems (partially or fully)? Did agile create new problems?
10. Describe your team: how many people, what are their roles. Are these people assigned solely to this project, or do they multitask on multiple projects? If the latter, what percent of their time is devoted to your project?
11. How complex is your software? What is the major source of this complexity (domain, software, management)?
12. How critical is your software and why?

Reminder: For any question, if the answer has changed over time, please describe how and why.

13. How accessible are your customers? Does your customer group have a single voice, that is, is there one person that provides the authority to make decisions when customers disagree with each other?
14. How familiar are your customers with the software development lifecycle?
15. How are features/issues prioritized? In what ways does this work well, and in what ways not so well?
16. How are requirements gathered and tracked? In what ways does this work well, and in what ways not so well?
17. How are completed issues accepted (that is, approved as complete)?
18. How do you estimate how long things will take? How does this vary for bugs versus planned scope? How successful are your estimates? How do you deal with issues that take longer than estimated?
19. Describe your planning processes. Describe ways that agile ideas do not apply to your planning situation.
20. Describe what sorts of planning problems you still have today. Describe hurdles your project environment presents to planning effectively.
21. How often do you release software to your users? Patches vs. larger deployments. Describe changes, as well as any mistakes you may have made.
22. Describe communication with scientists/other customers during: planning, development, acceptance, and maintenance.
23. How do you test your software? How do you protect against regression errors?
24. Do you have a QA process (beta testing) before releasing software? Describe the how and why.
25. What is your automated unit test coverage (tests per public method or percentage of statements executed or some other meaningful measure)? If you have automated functional/acceptance tests, what is the coverage like there?
26. How have your defect rates changed over time, and what are the possible reasons for the change?
27. Does your team have the resources it needs to be fully effective?
28. Does your team reflect on its effectiveness? How often? What changes have you made as a result of these reflection sessions?
29. How and how often do you review code/design/etc?
30. Describe your experience with pair programming, and why you don't do more of it (since we all don't do too much).

For the next five questions, discuss how each principle of agile software affected your project experiences.

31. Embracing Change
32. Collaboration
33. Technical Excellence – do not sacrifice technical excellent, it is just as important as delivering functionality and meeting deadlines
34. Simplicity – the "simplest thing that can possibly work"
35. Working Software – bias for working software over comprehensive documentation

36. Did you do anything "non-agile"? Why did you do it?

The interviewer also presented ad hoc follow-up questions during the interview to elucidate interesting issues encountered in the respondents' answers to the above questions.

Appendix

The following is the list of survey questions asked of each author. Each interview took approximately one hour. The interviewer presented the questions over the phone (except when he interviewed himself), so they are not recorded in sufficient detail to stand by themselves as a written interview. Please email the interviewer (moses@moseshohman.com) with any questions.

For any question, if the answer has changed over time, please describe how and why.

1. Describe your project.
2. Describe your users, customers, whatever you call them. If your software is intended for a public audience, describe your user/customer representatives.
3. Did you join the project at the beginning or was the project already in development before you?
4. How long have you been with your project? How long has it lasted overall?
5. When did you introduce agile methods?
6. What existed before the introduction of agile?
7. Were you the main proponent of agile, or were there others? If others, describe.
8. How did you learn about agile?
9. Why did you introduce agile, and in what ways did you introduce it? What problems did you hope agile principles would solve? Did you solve these problems (partially or fully)? Did agile create new problems?
10. Describe your team: how many people, what are their roles. Are these people assigned solely to this project, or do they multitask on multiple projects? If the latter, what percent of their time is devoted to your project?
11. How complex is your software? What is the major source of this complexity (domain, software, management)?
12. How critical is your software and why?

Reminder: For any question, if the answer has changed over time, please describe how and why.

13. How accessible are your customers? Does your customer group have a single voice, that is, is there one person that provides the authority to make decisions when customers disagree with each other?
14. How familiar are your customers with the software development lifecycle?
15. How are features/issues prioritized? In what ways does this work well, and in what ways not so well?
16. How are requirements gathered and tracked? In what ways does this work well, and in what ways not so well?
17. How are completed issues accepted (that is, approved as complete)?
18. How do you estimate how long things will take? How does this vary for bugs versus planned scope? How successful are your estimates? How do you deal with issues that take longer than estimated?
19. Describe your planning processes. Describe ways that agile ideas do not apply to your planning situation.
20. Describe what sorts of planning problems you still have today. Describe hurdles your project environment presents to planning effectively.
21. How often do you release software to your users? Patches vs. larger deployments. Describe changes, as well as any mistakes you may have made.
22. Describe communication with scientists/other customers during: planning, development, acceptance, and maintenance.
23. How do you test your software? How do you protect against regression errors?
24. Do you have a QA process (beta testing) before releasing software? Describe the how and why.
25. What is your automated unit test coverage (tests per public method or percentage of statements executed or some other meaningful measure)? If you have automated functional/acceptance tests, what is the coverage like there?
26. How have your defect rates changed over time, and what are the possible reasons for the change?
27. Does your team have the resources it needs to be fully effective?
28. Does your team reflect on its effectiveness? How often? What changes have you made as a result of these reflection sessions?
29. How and how often do you review code/design/etc?
30. Describe your experience with pair programming, and why you don't do more of it (since we all don't do too much).

For the next five questions, discuss how each principle of agile software affected your project experiences.

31. Embracing Change
32. Collaboration
33. Technical Excellence – do not sacrifice technical excellent, it is just as important as delivering functionality and meeting deadlines
34. Simplicity – the "simplest thing that can possibly work"
35. Working Software – bias for working software over comprehensive documentation

36. Did you do anything "non-agile"? Why did you do it?

The interviewer also presented ad hoc follow-up questions during the interview to elucidate interesting issues encountered in the respondents' answers to the above questions.

Back