An Agile Case Study

Using Agile for FileMaker Development Projects

Jason Mundok
jason@jasonmundok.com
www.jasonmundok.com
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>25</td>
</tr>
<tr>
<td>Preparing Test Cases</td>
<td>25</td>
</tr>
<tr>
<td>Running Test Cases</td>
<td>26</td>
</tr>
<tr>
<td>User Testing</td>
<td>26</td>
</tr>
<tr>
<td>More Iterations</td>
<td>27</td>
</tr>
<tr>
<td>Beta Testing &amp; Deployment</td>
<td>27</td>
</tr>
<tr>
<td>Conclusion</td>
<td>28</td>
</tr>
</tbody>
</table>
About the Author

I have been developing custom FileMaker projects since 2000. Like many of my colleagues, I was introduced to FileMaker as a tool to fix a particular problem that I faced at my job.

Working as an assistant to an event coordinator for a regional public school agency, I was tasked with scheduling and managing a one-day conference that included over a dozen sessions and approximately 500 attendees. Each attendee was allowed to prioritize their session choices and my task was to figure out the best way to configure the schedule based on various locations and timeslots, and assign attendees to their highest priority choices. FileMaker came to the rescue and I was immediately hooked.

In 2004, I joined the FileMaker development team at IT Solutions, a FileMaker Business Alliance Platinum Member based in Philadelphia, and I entered the world of FileMaker consulting. I became a certified FileMaker developer, an Authorized FileMaker Trainer, and eventually a full-time project manager. During my tenure at IT Solutions, we adopted an Agile software development methodology and crafted it to specifically fit in the FileMaker development paradigm.

Since early 2012, I have been an independent consultant offering custom FileMaker development services and project management training and coaching to a variety of FileMaker consulting shops around the United States.

Over the years, I have had the privilege of speaking about the topics of Agile software development in a FileMaker environment and using FileMaker to maximize workflow efficiencies at several FileMaker Developer Conferences, FileMaker user group meetings, and IT Solutions sponsored seminars in Philadelphia and New York City.

Introduction

A Look Back

In early 2006, I was working as a FileMaker developer and trainer for IT Solutions, a FileMaker Business Alliance Platinum member based in Philadelphia. At that time, we discovered a methodology for developing software that quickly became the standard process for all of our custom development projects; both FileMaker based and web applications written in .Net and PHP. The methodology was called Agile, a term that was relatively new in the software development world, and practically unheard of in the FileMaker development community.

At the time, we had a client coordinator and a handful of developers on staff, but nobody was specifically managing projects. Developers were self-managing some
relatively large projects and while some of them were satisfactory, many weren’t. Our biggest project management challenge was the inability to predict the outcome of projects throughout the development life cycle. We often didn’t know until the last minute whether we were going to come in for a smooth landing with a happy client, or smash into the side of a mountain. And the difference between those outcomes often had more to do with the behavior of our clients than our own project management. If clients were easy-going and flexible with their budgets and timelines, we would typically do well, otherwise we faced some daunting challenges that we were unable to control.

Our process was pretty standard for the time. We would engage a client in several sales and discovery meetings, collect as much information as we possibly could, and then generate a fairly comprehensive technical specification. These documents included drawings and diagrams, lists of fields, layouts, and scripts, and any other bits of technical information we could pull out of our discovery meetings. They were printed and put into a “project binder” that became the blueprint for the project and the basis for the time and cost estimates. Development of a technical specification and project estimate was an extremely time-intensive and labor-intensive exercise, the cost of which was often shared between the client and us.

After a project started, we would disappear for several weeks, follow the specification document and start developing. Juggling several projects and support or maintenance clients, developers often didn’t spend too much time on any particular project until closer to a milestone meeting. Meetings weren’t scheduled on a regular basis because we never really knew how much time we’d be able to spend on the project during a given week, and we didn’t want to participate in a review meeting after a month passed by with very little to show. So we’d wait as long as we could between meetings, dragging the projects out far longer than expected or necessary.

In the best-case scenarios, we scheduled a few evenly paced review meetings and were able to get a lot of work done between them. As the development phase for those projects came to a close, we implemented the new database system into a testing environment and hoped for the best. Sometimes we won, sometimes we lost, but we never really knew how things would play out until we heard back from the client. We were unable to predict the outcome.

This process was as close to a traditional software development life cycle as we were able to get. We acquired as much information up front as possible, expressed that information as a specification document, and then started working. We demanded change forms for any changes along the way, and hoped for the best when rolling out the software into beta testing.

We worked very hard to improve our implementation of this traditional approach, but after several less-than-successful projects, we knew there had to be a better way
to develop custom software projects. Then we discovered Agile software development.

**Agile Manifesto and Scrum**

In 2001, seventeen custom software professionals went skiing. Well, they actually held a summit to discuss the common ground between emerging practices in the software development, but they happened to do it at a ski resort in Utah. These practices were moving away from the traditional development life cycle and were focused more on adapting to change and cooperative relationships between all members of a development team. There was no formalization of a single Agile methodology, but rather a recognition of several emerging methodologies that now fall under the Agile umbrella, such as Scrum, Extreme Programming (XP), and the Unified Process (UP). (Larman, 2004).

As a result of the summit, the group composed the *Manifesto for Agile Software Development* that defined agreed-upon values of the group (Larman, 2004). The manifesto is published on the Internet at www.agilemanifesto.org. The *Principles behind the Agile Manifesto*, which is a list of 12 underlying principles, the manifesto history, and a list of signatories, or software development professionals, who agree with the values and principles behind the manifesto, are also included on the website (Cunningham, 2001).

Upon my introduction to Agile in 2006, I became most familiar with Scrum, which is an approach that originally described a new product development strategy by Takeuchi and Nonaka in 1986. The term “Scrum” comes from the sport of rugby and refers to a strategy where team members pass the ball back and forth as they advance up the field.

Jeff Sutherland began using this approach for software teams at Easel Corporation in the mid-1990s. At the same time, Ken Schwaber was also using a similar approach for development at Advanced Development Methods. Sutherland and Schwaber formalized the process in 1996 (Schwaber & Beedle, 2002).

Scrum is a team-based approach to software development. It focuses on meeting business needs and fostering cooperation. Complex and very sophisticated systems are developed incrementally, based on the team’s skills and knowledge, rather than through formal complex system plans (Schwaber & Beedle, 2002).

Scrum seemed like a good fit for the FileMaker development projects we were working on at IT Solutions. Each custom project was essentially a brand new product, something that never existed before. As developers we were essentially engaging with our clients in new product development.
Making Scrum Fit

Like other Agile methodologies, Scrum is primarily being used in enterprise-level in-house development environments, where one or more seven-developer teams are working on a single system over an extended timeline. FileMaker projects are typically much smaller and can often be made up of a single self-managing developer, so while Scrum can make a lot of sense for these projects, the process must be tweaked to apply the principles to a FileMaker environment.

After one year and a handful of very successful projects using a modified flavor of Agile, I began evangelizing the methodology in the FileMaker community. Since then, I have accumulated seven years of experience managing several dozen Agile projects. During most of that time, I was fortunate to work in a FileMaker development shop where I could act as a dedicated project manager working alongside a handful of very talented developers and a very thorough and dedicated software tester. It was an ideal situation.

Over the years I’ve often been asked whether or not the methodology could also be used for a single-developer project with no other resources. While it’s not the ideal scenario, it is very possible and can prove to be extremely helpful for a solo developer to stay in control of large and complicated projects.

While the purpose of this paper is not to prove the viability of using Agile in a single-developer project, I will be using a case study as an example to demonstrate the use of the methodology. The case study is from a recent custom FileMaker development project that was estimated, developed, and managed using the Agile methodology without any additional resources. The FileMaker solution that was created during the project is called Impact. References to the Impact project throughout this paper will be colored blue.

Project Overview

During the summer of 2012 I received a call from the CEO of Innovative Hospitality Solutions (IHS), a food service management consulting firm based in Lewes, DE. IHS is a small company and the CEO happens to be my brother, Matt Mundok. We had never done any professional work together, and while we have a great relationship, he wasn’t quite sure if I could help him solve his data collection and management challenges.

IHS has a handful of consultants who perform onsite evaluations of corporate, campus, and healthcare dining facilities. Third-party vendors operate these dining facilities, so IHS conducts evaluations of both the facilities and the vendors. Then a comprehensive executive report is delivered to the organization’s leadership team.
IHS had been collecting information for these evaluations on paper and then keying the results into lofty Excel spreadsheets. Executive reports were manually created in the spreadsheets and all of the data were completely disconnected from other data sources. IHS was interested in exploring ways to collect the onsite data using iPads and then merging the data in a system at the company headquarters where the executive report could be generated automatically. FileMaker Go was the perfect platform for this challenge and we began to explore ideas to create the new FileMaker solution called Impact (see Figure 1).

**Figure 1**

**Roles**

There are several roles that need to be played during the course of an Agile development project. The five primary roles are **project manager**, **developer**, and **tester** on the creator side of the project, and **project owner** and **subject matter expert** on the consumer side. Here’s a brief overview of each role.

**Project Manager**

The primary responsibility of the project manager is to ensure that everyone on the team is aware of and comfortable with the project process. Because we are applying this methodology to relatively small custom development projects, we are often working with consumers who have never been part of a software development team. It’s critical that all team members are educated about how the process works, which can sometimes require consistent reminders throughout the life of the project.
A secondary, but also critical responsibility for the project manager is to remove roadblocks that stand in the way of developers and testers so that they may continue their work at the planned velocity. For example, meeting scheduling, budget and status reports, or purchasing third party technology are all tasks that can take up a developer’s valuable time and energy. The project manager should take responsibility for such tasks and/or delegate them to an appropriate third party if one exists within the development organization.

Developers

The primary responsibility of the developer is to develop the software as outlined on the feature list at the appropriate velocity, and as planned at the project’s iteration meetings. Developers are also responsible for managing the detailed requirements gathering sessions during iteration planning and then demonstrating completed features during the iteration review.

The project manager is not a technical resource and therefore should not lead the review and planning sessions, even if he or she has a technical background. Only the developer can know whether or not enough requirements have been gathered to move forward with a feature, and only a developer can properly demonstrate a completed feature during the review.

Developers are also responsible for composing iteration test cases so that the software can be properly tested before it is reviewed.

Testers

The primary responsibility of the tester is to review and run the project test cases as provided by the developer, and then report any defects found in the software to the developer. The tester should be reviewing the software from an end user perspective, not a technical perspective. The tester should never attempt to fix the software during testing, as they may not have an understanding of the effect a change may have on another part of the system.

Testers may also provide suggestions to the developer regarding user experience, even if those suggestions fall outside of the test case. Extra user experience feedback should only be considered if it is covered by the project feature list and fits within the project budget.

Project Owner

The primary responsibility of the project owner is to represent the vision and goals of the project for all eventual end users. Through this representation, project owners are responsible for making critical decisions about the direction of the project and delivering all necessary requirements either personally, or through the help of subject matter experts.
Logistical responsibilities for the project owner include meeting scheduling, resource scheduling, the delivery of promised information, and consumer-side testing.

The person playing this role should be able to immediately respond to 80% of the questions posed by the creators, and should be able to find any additional information within a reasonable amount of time, usually 24 hours. As a result, the project owner needs to have decision-making authority.

**Subject Matter Experts**

The primary responsibility of a subject matter expert (SME) is to provide requested requirements to the development team through iteration planning meetings or ad hoc requests during an iteration. A SME representative should be present at all review and planning meetings that are relevant to their day-to-day functions and should be available for software testing throughout the project.

These project roles do not necessarily represent specific people. While it is ideal to have a different person, or people, representing each of these roles throughout a project, sometimes that is not possible. At the very least, you need one person to represent the creator roles and another to represent the consumer roles.

The Impact project was conducted with only two people, myself playing the roles of project manager, developer, and tester, and Matt playing the roles of project owner, subject matter expert, and tester. With only two of us involved in the project, it was critical to remember every responsibility that each role represented to ensure delivery of a complete and successful project.

**Feature Planning**

Agile development projects begin with feature planning, which is similar to a traditional discovery phase. The key difference is that the outcome of feature planning is a list of user-based features that describes what users will be able to do with the system or expect the system to do automatically upon completion of the project.

Feature planning is essentially lightweight discovery that is not intended to define detailed technical specifications. In fact, no technical decisions are typically made during feature planning. Instead, the consumers of the software will talk through all relevant business processes, demonstrate current business applications or manual systems used to manage those business processes, and describe any new desired functionality that doesn't exist already. The development team captures that information in language that describes the functionality from the perspective of the end users.
For the Impact project, Matt and I did a couple 2-3 hour feature planning videoconferences and screen shares. First he explained the process of going onsite and conducting evaluations of food service facilities on the campuses of IHS clients. Then, after I had an understanding of the evaluation process, he shared the spreadsheets that were used to key in their onsite data and the executive reports that were created using that data.

While we worked through the business process and spreadsheet, I took notes and wrote draft user-based features. I would occasionally interrupt with questions and clarifications as needed. Most of the information I sought was how an end-user would actually use the system, and what automation that user would expect. I wasn’t interested in documenting anything technical, as I wasn’t really making any technical development decisions at the time, but admittedly some FileMaker development techniques that I would eventually use became obvious to me even at that early stage of planning.

The Feature List

A feature list is the result of the feature planning meetings and discussions. It is simply a compilation of all expected functionality that will be included in the proposed system. Each feature in the list is estimated individually in hours. The sum of those features makes up the estimated development time, but doesn’t include any additional supplemental costs, such as time that will be spent in meetings, managing the project, or testing the software.

Here are some examples of features that were created as the result of our feature planning meetings for the Impact project:

- User can manage client records on a form view that will include demographic information and contact information for a single client contact. This is not intended to function as a full CRM but could be expanded later.
- User can add, edit, delete, or hide locations from the client form view. Locations will not have a separate interface and will include a few fields such as location name, campus, etc.
- User can see evaluation categories on a list view, select a category and drill down to see category details on a form view.

You can see these features in the context of the actual feature list in Figure 2.
Figure 2

Estimating

During Agile presentations and workshops over the years, I have often been asked for a formula to accurately estimate the features for an Agile FileMaker development project. Since every developer is different, and every development team contains a different combination of talent, it’s practically impossible to offer a one size fits all formula for estimating features, but here are a few techniques that I have found useful over the years.

Pick one and make everything else relative.

Many Agile advocates estimate in points rather than hours. One technique that is often used assigns a point value to one of the features in the list that is easy to estimate. Then, another feature is compared to the first feature to determine if it is more or less complex and to what degree. Each un-estimated feature is compared to previously estimated features to determine its relative value.

While I typically estimate in hours rather than points, I frequently use this comparative method to question whether an unknown feature would take more or less time to develop than a feature that represents common, or known functionality. We often build very similar features across multiple custom FileMaker solutions, so it is helpful to start by estimating features that are very similar to work that has been done before, and save less known features for later in the process.

Estimate in days.

Typically FileMaker features should be detailed enough to represent somewhere between 2 and 12 hours of development time. If a feature represents more than 12 hours of work, it is probably several features being described as one. If the estimate is less than 2 hours, that functionality could probably be rolled up into another feature. The 2 to 12 hour range represents approximately a third of a day to two days of development.
When thinking about how long it should take to develop a feature, I often question whether or not I could start and complete the feature in a single day if given all of the necessary information. I imagine spending an entire day on that single feature and try to determine if I would have any trouble completing the functionality in that time frame.

If I think it would take less time, I ask the same question about a half day. If this feature was the first thing I started in the morning, could I finish developing it by lunch or could I see myself having to work on it during the afternoon? Or does this feature seem daunting enough that I don’t feel that I could finish it, even with a whole dedicated day of development? Using a day as the unit, or the time before or after lunch can sometimes be easier than thinking about the act of developing in hours.

After determining how long it should take in terms of days, it’s easy to convert that time into hours using two-hour increments. A whole day would translate to 6-8 hours and a half-day to 3-4 hours. A feature in between would land around 5 hours.

**Analyze the past.**

By far the best way to accurately estimate development time is to simply try it, and then keep track of how long the development actually took. After writing multiple feature lists, you will be able to use past experience as a baseline from which to estimate new features. Ask yourself how similar a feature is to something you have developed in the past, and then start estimating with the actual time it took to develop the previous feature. New challenges can be compared with known features using the comparative method explained above.

**Other factors.**

Other factors may also need to be taken into consideration when estimating. If the feature is simply a replication of functionality that exists in an older system, then it is fairly well defined and may be easy to accurately estimate and develop. If you are creating brand new functionality, it may require a prototype or several rounds of feedback to refine the feature to the satisfaction of the consumers. Therefore, some additional time should be added to the feature estimate.

Another important factor to consider is the organization level of the consumers. How well did they accurately convey their needs during the feature planning process? Do they need to refine their own business processes before they can effectively describe the desired functionality? The behavior and communication style of the consumers during feature planning should be an indicator of whether or not additional time should be added to feature estimates to compensate for potential refactoring during the development process.

If you are estimating for other developers, you may also need to take into consideration which of those developers will actually be creating the software, if it is
known at the time of the estimate. What are their strengths and weaknesses? How much experience do they have developing similar solutions? How well do they receive and accurately understand information? You can adjust your baseline estimates according to those skillsets.

There are many variables that need to be considered when estimating how long a feature will take to be developed. It is impossible to provide a generic formula that will for anyone but my own team, and only then after plenty of experience working with and getting comfortable with the style of each individual on that team.

Supplementals

Before you can deliver an actual estimate, you also need to account for a number of additional costs that support the actual development of the software. These costs are supplemental to the development process and should be estimated and tracked separately to ensure that there is plenty of time for them specifically outlined in the budget.

Supplemental categories may vary from project to project, but here are some common ones used in almost all projects.

- Meetings – the Agile process is very communication intensive, including regularly scheduled review and planning meetings, along with additional chatter as needed during the iterations.
- Project Management – the project manager is responsible for keeping everyone on the team educated about the development process, removing roadblocks that prevent developers from making progress, scheduling meetings, and keeping everyone up to date on the project status.
- Testing – the software should be thoroughly tested during each iteration and then again prior to deployment.
- Deployment – developers need to eventually move the new software into production. They may need to assist with FileMaker Server installation and configuration, FileMaker Pro client installations, or assistance with initial connections from desktop computers or mobile devices.

In my past experience, these supplemental costs typically make up 40% - 50% of the total development costs. For example, a project estimated at 200 development hours would include an additional 80-100 hours of supplemental time.

Total development hours for the Impact project was 64 hours, with 22 hours in supplemental costs (see Figure 3). That is slightly lower than 40%, but because I was playing the role of both the developer and project manager, I could eliminate some of the hours that would typically be used coaching the process and removing developer roadblocks. Had I been handing the project off to another developer, I would have included additional hours for project management and meeting time.
Release Planning

Upon acceptance of the proposed feature list and project estimate, it's time to get to work. The next step of the process is release planning, or mapping out how the project will unfold across the estimated timeline. This takes place during the first meeting of the actual project.

Releases are four to six week time periods within the overall project timeline. At the end of each release, a copy of the system is distributed to the consumers for testing and review. The end of the final release marks the end of development and the beginning of beta testing.

In preparation for release planning, the project manager will review the feature list to get a general sense of the order in which feature development should occur and to clarify the amount of development time estimated for each week. He or she should also collect as much information as possible regarding potential timeline roadblocks for developers and testers, such as planned time off or commitments to other projects. Having a solid understanding of this information will help the project manager lead the initial meeting and set realistic timeline expectations.

The release-planning portion of the initial meeting starts with a presentation by the project manager about the Agile methodology and project process, including an overview of roles and responsibilities, the iteration review and planning process, and the proposed timeline.

Following the project process overview, the project manager schedules iteration meetings (iterations are discussed in the next section) with the team, establishes communication channels for the project, and sets expectations for a project timeline.

Iteration meetings are scheduled as consistently as possible on the same day and time every two weeks for the proposed life of the project. It is important to establish and confirm the entire project meeting schedule during release planning to avoid conflicts and scheduling challenges later in the project. The project manager should send all review and planning meeting invitations to the team immediately following

© 2013 Jason Mundok LLC
the initial project meeting to ensure established agreed upon deadlines and milestones.

Communication priorities and styles can vary from project to project. Project managers need to establish the most efficient way for the team to communicate throughout the project by asking questions like: who should be copied on project emails? Who should receive budget and status updates? What is the best channel for communication based on the urgency of the request? A short conversation about communication during release planning will avoid confusion later in the project.

The project timeline can then be estimated and presented based on the agreed upon meeting schedule, team availability, and the estimated amount of dedicated development time per week. The project manager should also allow for an additional two weeks for beta testing at the end of the estimated project timeline. After a timeline and projected delivery date are presented, the project manager should be sure to inform the team that the delivery date is only an estimate and unforeseen circumstances may, and probably will affect the projections.

During release planning for the Impact project, we established a consistent meeting schedule based on my available development hours each week. I had planned on dedicating about 10 development hours per week to this project, which translated to a six-week time period, broken down into three iterations. We planned our meetings on Friday morning every two weeks for a total of four meetings, including the initial planning meeting. Having a consistent time and day to meet meant that we didn’t have to face the challenge of finding common available meeting times for review and planning sessions at the end of each iteration, which can even be difficult for only two people.

We also established that quick questions and clarifications should be asked via text messaging due to Matt’s busy travel schedule and heavily trafficked email inbox. Any communication that didn’t require an immediate response could be emailed, and unscheduled phone calls were completely ruled out unless a text message led to the need for more direct communication.

**Iterations**

The Agile methodology calls for all work to occur during iterations, or cycles of time. For FileMaker projects, I have found that two weeks is an appropriate amount of time for a development cycle based on the amount of work that can be reviewed and planned during a two or three hour meeting.

An iteration is like an abbreviated traditional software development life cycle in that it starts with a discovery phase (the planning part of the iteration meeting), followed by development, testing, and review (the review part of the iteration meeting). The purpose of breaking down work into short iterations is to keep
planning information as fresh and relevant as possible. Developers should only ever be working on features that were planned in detail at the beginning of the current iteration. The goal of the developer for a given iteration should be to develop and complete all features that were planned for that iteration. Those features should also be tested before each review meeting to ensure that only working software is shared with the team during review.

Iteration meetings contain planning, review and a retrospective.

**Iteration Planning**

The purpose of the planning portion of regularly scheduled iteration meetings is to gather detailed requirements for the features that are scheduled for that iteration, and only those features. Planning and discussion for all other features should be deferred to a later time, when they are scheduled for development. Time should never be wasted during iteration planning to discuss a feature that is not on that iteration’s schedule unless it is directly related to one that is, and will somehow affect its development. As a result, time is never wasted planning or developing features that may be removed from the feature list at a later time due to other, higher priorities that may not yet be known.

While features are discussed and documented during feature planning, those discussions should be kept at a fairly high-level and will not yield enough information to make design decisions or accurately develop the features. More detailed discussions are saved for iteration planning, directly before development begins.

The requirements gathering during iteration planning should be as detailed and granular as possible. A comprehensive list of details about each feature should be discussed. For example, if one of the features is a report that the new system will generate, then attributes such as the formatting, header, footer, fields, and required calculations should all be discussed during the planning session. The team should review any existing versions of the report and discuss any desired modifications, or if the report is new, map out all specifications.

The developer’s goal should be to walk away from iteration planning with enough information to develop all scheduled features without any further interaction. Of course, should any question come up during the iteration, the developer should immediately reach out to the project owner or SMEs through the agreed upon communication channels. Developers should never make assumptions about features without checking with the consumers.

For the Impact project, we planned approximately 20 hours of development during iteration planning meetings. That allowed for plenty of time to discuss features at a very detailed level during our sessions without getting bogged down in long meetings.
As we began our planning, because the first few features included user-facing layouts, one of the first items to discuss was layout design, including a choice of FileMaker themes (see figure 4) for the project.

Agile requires that only the currently scheduled features are discussed during planning, therefore if none of the initial features included a user-facing layout, then a choice of FileMaker themes would not have been necessary until a later iteration.

Figure 4

Iteration Review

Approximately two-weeks following an iteration meeting, the team will reconvene for the next meeting, which begins with a review of the development that occurred during the iteration. All features that are reviewed should be completed and tested by the developer(s) and tester(s). The team should be informed if any of the scheduled features was not completed and tested prior to the meeting, and those features should be rescheduled for the next iteration.
The iteration review is a demonstration by the developer of all new functionality that was worked on since the last iteration meeting. The feature list should be used as a guide for this demonstration so that all functionality can be tied directly back to the satisfaction of one or more features.

Whenever possible the end users, the project owner or subject matter experts, should drive the software during the demonstration. This will provide an immediate view into the instincts and behavior of how users will naturally approach the system. It will also begin to transfer a sense of ownership of the system from the creators to the consumers.

The project owner is responsible for accepting or rejecting each feature during the review. If the feature is completed and demonstrated in such a way that it fulfills the original feature, then it should be accepted. If for any reason more work is needed to accept a feature, then it should be rejected and queued up for the next iteration. Additional planning for the feature can then be conducted during the planning portion of the meeting.

When a feature is accepted, there is acknowledgement by the team that it has been satisfied, but the part of that system that the feature represents may need to be worked on again at a later time if required by another feature. For example, a form view layout may require an additional tab and new portal based on a feature that is added in a subsequent iteration. Whatever is needed to satisfy a feature should be included, but only during that feature’s scheduled iteration.

Because the developer should only be planning, developing and reviewing the features that are scheduled for the current iteration, the system may appear thin during the first one or two iteration review sessions. That’s perfectly acceptable as long as the features that are demonstrated accurately reflect what was planned.

Developers should always avoid the tendency to work ahead and develop functionality that they assume will be needed. The Agile methodology allows for negotiated changes in the feature list throughout the project if the consumer’s priorities or business processes change. Therefore, any work that is done ahead of iteration planning could prove wasteful and irrelevant.

The Impact project was broken into three sections for each of the three iterations: the desktop interface, the mobile interface, and reports. Planning for the first iteration focused solely on the desktop interface. We deferred all discussion about the mobile interface and reports until the second and third iteration planning sessions. Therefore, the first review meeting was a demonstration of the desktop interface only (see figure 5).

Because the project was relatively small, the desktop interface was essentially complete and mostly accepted during the first iteration review. There was some feedback that caused a few of the features to be rescheduled for the next iteration,
but for the most part that portion of the project was finished and planning for the mobile interface was started.

**Figure 5**

**Enhancements**

Iteration review is when the creative juices really start flowing for a project team. During the feature planning, release planning, and initial iteration planning, everything about the project is theory and speculation. The first iteration review meeting is when the project becomes a reality and consumers can actually see the potential for the system. This is a critical moment for the project manager to reinforce the project process and remind the team that the entire project (scope, budget, and timeline) hinges on staying true to the existing and agreed upon feature list.

Feedback from consumers during iteration review meetings is often not reflected on the initial feature list. The project manager must have the confidence to determine what should or should not be considered a modification to an existing feature or what should be considered an enhancement to the original feature list.
The Agile methodology allows for enhancements, or new feature requests, to be considered. They are actually encouraged if they ultimately support the creation of the most accurate software possible to meet the needs of the consumers. To be included in the project, however, the project manager and project owner must negotiate how their inclusion will affect both the timeline and the budget of the project.

The most effective way that I have found to deal with enhancements is to establish a separate enhancements list at the outset of the project, and to inform the team that we will use it as a place to log any new ideas that fall outside of the current feature list. As items are added to that enhancements list, a priority can be assigned to it to determine how important it is to the current project.

If an enhancement is critical to the success of the project, then it can be included if a feature with a comparable time estimate is removed, or if the consumer agrees to increase the budget and alter the project delivery date. It is not uncommon for consumers to change priorities mid-project, adding new features to the project and deeming other features unnecessary. Feature swapping can be a very effective way of developing a collaborative spirit for the project team, while maintaining control of the budget and timeline. It also reinforces the goal of delivering software that accurately reflects the needs of the consumers regardless of what was originally planned.

Enhancements that are requested but are not critical to the success of the project can be tracked on the enhancements list. They may still be included later in the project if there is a budget surplus and the development will not have a negative affect on the project timeline. Any enhancement that is not included in the project can be considered for development in a future phase, or through ongoing post-deployment support.

**Retrospective**

At some point during the iteration meeting, the project manager should facilitate a short retrospective discussion about the project to be sure that all team members are comfortable with the process and their responsibilities within it, and that all project goals are being met. If there are any issues or roadblocks that are preventing goals from being met, the retrospective offers a time to address them or plan additional time to meet and discuss them with the appropriate parties.

The retrospective is also a time when the project manager presents a status update on the feature list, budget, and timeline. (see Figure 6 for a budget report from the Impact project) There are several elements that should be included in each status update:

- Feature list status – a review of how many and which features have been completed and accepted to date.
• Budget status – a review of how many actual hours have been used so far in each category and how they compare with the development that has been completed and accepted.
• Budget burn down – a comparison of remaining budget hours versus the remaining estimated development and supplemental hours.
• Timeline status – a review of how close a project is to the original timeline and when completion of the project is projected.
• Projected issues – a report of any projected issues for the project in terms of scope, budget or timeline.

![FileMaker Development Project Status Update]

![Hours Estimated vs. Actual]

Figure 6

During the Impact project, retrospectives were very short discussions because the entire development team only included Matt and me. The project process was explained and understood at the beginning of the project, and we weren't having any trouble communicating.

We were never at any risk of going over budget during the project. I kept track of my development time at the feature level to test my own estimates and was coming in low fairly consistently. A quick overview during the iteration meeting with a review
of the estimates versus actuals was all that was needed to convey the feature and budget status.

However, our calendar timeline shifted considerably. Unexpected travel caused one of our review meetings to be postponed, which altered our delivery schedule. During the retrospective we discussed how this would affect the timeline, and we agreed on a revised delivery date.

Even when things are going well, it’s important to at least take the time to acknowledge that a process is still being followed and be sure that everyone on the project team is clear about next steps. These discussions set the stage for open communication if problems arise later in the project.

**Development**

**Tasking**

Following iteration planning meetings, the developer should compile all notes taken during the session and write out tasks for each feature. Since the features have been discussed in detail, freshly written tasks will represent the most accurate possible description of how the developer should proceed. The tasks become a technical to-do list that the developer can refer to throughout the iteration.

Next, the developer should estimate each task to determine whether or not it will be possible to complete the planned development work within the iteration. If it is discovered that development will likely take longer than anticipated, the developer can reschedule features for the following iteration. If task estimates reveal that the planned development will take less time than planned, additional features can be added to the current iteration as long as there is time to gather requirements with the project owner and/or subject matter experts at some point mid-iteration. In either case, the entire project team should be briefed on any changes with the iteration feature schedule.

I have found task writing to be a very effective way of planning for iteration development, especially when bouncing between different projects and clients. Returning to a well-written task list will allow a developer to jump right back into development.

With that said, tasking might not always be necessary. If features are fairly simple and are very well defined based on meeting notes and supporting documentation, then spending the extra time to write out tasks could prove wasteful. The decision to write tasks can be left to the discretion of the developer as a useful tool when planning complex features, or if the developer is struggling with completing scheduled features within the iterations.
By the time development actually occurs during the iterations, there should be little question as to what needs to be done to fulfill each feature. Because the features are scheduled into iterations based on available hours and feature estimates, expectations of what will be reviewed at the next meeting should not exceed a reasonable amount of work. Additionally, either detailed meeting notes for the scheduled features or tasks written immediately following the planning session should be at the developer’s disposal when the actual development begins.

If the developer has any question about a particular feature, he or she has open and established communication channels with the project owner and SMEs to reach out for clarifications. Developers should never make any assumptions about system requirements, business process, or workflow.

It is an ideal environment for developing software that accurately reflects the planned and agreed upon feature list.

**Dedicated Development Hours**

A critical factor in keeping a project on time and on budget is dedicating a consistent amount of time per week for development. All timeline and budget estimates are based on this commitment, and any deviation from it will affect the overall project.

Larger development shops often have the luxury of dedicating resources to a single project at a time, which can dramatically increase project velocity and shorten project timelines. In smaller shops, the demands of supporting existing clients and juggling incoming projects can make it very difficult to be dedicated to a single project. Solo developers have the most difficult challenges when it comes to dedicating consistent time to a project, but the process requires it to determine the appropriate number of iterations in the project timeline. The number of iterations directly informs time estimates for meetings, project management, and testing. Even if it can only be a few hours per week, carving out a set amount of time for development allows honest expectations about the budget and timeline to be set.

For the Impact project, I dedicated 10 hours per week, or 20 hours per iteration, for development and I actually scheduled those hours on my calendar on the same days and times each week. It was enough time to complete the project within a few iterations but it still left enough time in my week to address other projects that I was working on and other clients that I was supporting.

I have found 10 hours per week to be a very reasonable amount of time to dedicate to a single project when acting as a solo developer on relatively small projects of less than 120 development hours.
Testing

There are three levels of testing for FileMaker projects using the Agile methodology. Each feature is tested as the developer prepares one or more test cases for the feature. An external tester then performs the test cases. Finally, the project owner and subject matter experts have opportunities to test the software at each release, and then again during the beta testing period at the end of the project.

Preparing Test Cases

Test cases are a set of user instructions that describe how to use the database to satisfy the functionality as described in the feature. Test cases should always include a list of actions that a user should perform along with the expected result of that action (see figure 7). Optional elements of a test case include screen shots, sample data so that the tester can create a more authentic environment, and multiple accounts and passwords needed to login and test the system from different perspectives, depending on the complexity of the system’s security settings.

<table>
<thead>
<tr>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feature:</strong> User can add, edit, delete, or hide locations from the client form view. Locations will not have a separate interface and will include a few fields such as location name, campus, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Result</th>
<th>Verdict</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login to the database with a test user account (account: user1 password: test)</td>
<td>The client list view should be displayed.</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>Click on the blue drilldown arrow to navigate to the client form view for any client.</td>
<td>The desired client form view will be displayed.</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>Navigate to the Locations tab and add a new related location by typing in the blank portal row at the bottom of the portal.</td>
<td>A new portal row will be automatically added and the active checkbox will be checked by default.</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>Enter a campus name and a location type in the appropriate fields and click out of the portal.</td>
<td>The values will be saved in the portal row and the record will be committed.</td>
<td>Fail</td>
<td>The type field is not editable.</td>
</tr>
<tr>
<td>Modify the values in all fields in a portal row and click out of the portal.</td>
<td>The values will be edited and the record will be committed.</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>Toggle the active checkbox off and on.</td>
<td>Verify that the checkbox can be toggled.</td>
<td>Fail</td>
<td>The active checkbox is not editable.</td>
</tr>
<tr>
<td>Add two more locations in the portal and then use the delete button to remove one. Click out of the portal.</td>
<td>Verify that the desired location is removed from the portal and the record is committed.</td>
<td>Pass</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7

With the advent of easy-to-use video screen capturing tools such as Snagit from TechSmith, short videos that provide instructions on how to perform each feature’s functionality can alternately be created. Using videos cuts down on the amount of
time needed to compose test cases and can provide a much richer experience for the tester.

**Running Test Cases**

Testers should run the test cases exactly as the developer has written or prepared them. If any step of the test case does not pass as described, then the tester should reject the test case and flag it for revision.

A tester should never attempt to fix problems found when running test cases, even if he or she has the technical capacity to do so. There may be dependencies that are not obvious when running the test case that could break after making uninformed modifications to the system.

**User Testing**

At the end of each release in the project timeline, a copy of the system will be made available to the project manager and subject matter experts for internal testing. The testing environment should be as authentic as possible. If the system will eventually be hosted using FileMaker Server, then the test system should be hosted on a server and accessed exactly the same way the production system will eventually be accessed. Test accounts should be set up and distributed to the testers that are set to the same privilege sets that will be used in production. If possible and applicable, authentic data should also be loaded into the system prior to testing.

As users generate feedback items, they should be collected in a list that is similar to the feature list, but tracked separately. Bugs or defects found in the software should be immediately scheduled into subsequent iterations and billed to the testing hours.

Inevitably, as users begin testing the system, new ideas about additional functionality will emerge. These enhancements or changes to the system that fall outside of the original feature list should be tracked on the project enhancement list for future consideration.

As discussed in the iteration review section, if the requested enhancement is critical to the success of the project, then it can be added to the feature list, so long as a comparable feature is moved off of the feature list or the appropriate amount of estimated time is also added to the project budget, including additional time needed for supplemental costs caused by an increase in development hours.

For the Impact project, Matt acted as both the internal and external tester. I provided instructions on how to use the system and he provided the necessary feedback to address bugs and to log enhancements that could be added to make the system even better. Ideally, it is much better to have an internal tester who is not the consumer to get an additional perspective on the usability of the solution, but solo developers do not always have that luxury. The project owner or other
consumers can be used as testers as long as clear expectations about the state of solution are expressed.

At the end of the project, we had a surplus of hours to work with, so we were able to include all requested enhancements within the project budget.

More Iterations

As the project unfolds, the iteration pattern of plan, develop, test, and review continues until all of the features have been developed.

Projects take on their own personality based on the chemistry of the team and the unique set of required features. Sometimes velocity increases as the team gets more comfortable with each other and the process, and as the consumers get more familiar with the system’s potential. Other times projects slow down as more and more complex features get added. There’s a natural ebb and flow to Agile projects that is difficult to predict but finds a balance when executed properly.

Beta Testing & Deployment

When all of the project features have been developed, tested internally, reviewed, and accepted, then a beta version of the system is released to the consumers for final beta testing. The length of the beta testing period varies depending on the size and complexity of the project, but two weeks is a standard length that I’ve used successfully as a baseline. If the project spans more than one release, the consumers should have already received a test copy of the system and already done preliminary rounds of testing after each release. Therefore, beta testing should only be used to test the final few iterations of development and to introduce the system to a larger test pool within the organization.

As with release testing, user feedback should be organized as bugs or enhancements. Bugs should be addressed immediately. Enhancements should continue to be logged on the enhancements list and added into the project if the budget allows. Otherwise, they should be considered for a future development or support phase.

After all parties are satisfied with the accuracy of the system, it should be deployed as required in the original planning. Deployment of FileMaker projects happens in a variety of configurations and is project specific.
Conclusion

After seven years of experience managing FileMaker projects of all shapes and sizes using the Agile methodology described in this paper, I have learned that the process is by no means a magic solution that will guarantee successful projects. There are simply too many factors that can affect the outcome.

The Agile methodology is, however, an excellent tool that allows teams to predict the health of a project throughout the entire timeline. The ability to predict when a project isn't going well allows teams to make immediate decisions for course correction and ensure that everything possible will be done to collaboratively revitalize the health of the project. That is a powerful tool for success.

A year after finishing the Impact project, it has proven to be an effective solution for IHS to streamline its operations. Using FileMaker Go on the iPad (see Figure 8) for data collection isn't just a powerful tool for IHS operations, it's also an effective tool for IHS sales, demonstrating a well-thought out evaluation and data collection strategy.

Even with only two participants on the project team, the Agile methodology proved to be an effective process for collecting requirements, executing development, and deploying the Impact solution into production for the IHS team. By following the process guidelines, we had clearly articulated requirements, defined project deadlines, and well-executed development that led to a successful deployment and set the stage for ongoing improvements to the solution in future phases.
Figure 8