

**State of Michigan  
(SOM)**

# **SUITE Agile Process Guide**

**Version 1.0**

**Another Companion to the  
Systems Engineering Methodology (SEM) of the  
State Unified Information Technology Environment (SUITE)**

**July 2012**



**Department of Technology, Management & Budget**



**PREFACE**

It must be noted that Agile is really about collaboration, communication, and continuous improvement; if your team (including the customer) is not committed to these three items at the onset, Agile probably will NOT succeed. Many of the best practices associated with Agile have been successful in the public and private sectors over the past decade, and potentially have much to offer the DTMB.

Also to be noted is that Agile is NOT a methodology! It is simply a list of four statements (The Agile Manifesto) and twelve supporting sentences (Principles of Agile Software). Users may apply best practices associated with Agile (e.g., sprint cycles or paired programming) to improve their team’s existing methodology. This may sound like a nuance, but is actually an important distinction. For the purposes of this guidebook, the term ‘Agile’ is used as a shortened version of “collection of best practices often associated with (but not exclusive to) Agile.”

The purpose of this manual is to explain how to move forward using Agile with the SEM process. Some background material is provided on both Agile and SEM, but it is assumed that the reader has prior knowledge of both. The DTMB Agile Process Action Team (PAT) can be used as a resource for teams interested in learning more about Agile.

This process guide has been developed as part of a continuing effort to improve the quality, performance, and productivity of State of Michigan information systems.

**ACKNOWLEDGEMENTS**

The State of Michigan would like to thank the following individuals and organizations that made this initial version possible. Without their collaboration and information sharing, this would not have been achieved:

<b>INITIAL RELEASE (July 2012)</b>	
DTMB Agile Process Action Team	John Carey (lead), Don Carne, Bryan Farr, George Hamel, Tom Hudson, Alex Inglis Rosemary Klodt, , Tina Symington
Greater Lansing .NET User Group (GLUGNET)	<a href="http://www.glugnet.org">http://www.glugnet.org</a>
Mid-Michigan Agile Group	<a href="http://www.meetup.com/Mid-Michigan-Agile-Group/">http://www.meetup.com/Mid-Michigan-Agile-Group/</a>
DTMB Agency Services EPMO	Virginia Hambric

The following information is used to control and track modifications to this document.

<b>Revision Date</b>	<b>Author(s)</b>	<b>Section(s)</b>	<b>Summary</b>

## Table of Contents

<b>Chapter 1.0 INTRODUCTION AND BACKGROUND .....</b>	<b>1</b>
Why Agile, and Why Now?.....	1
ITIL and CMMI .....	1
<b>Chapter 2.0 SUITE SUMMARY .....</b>	<b>3</b>
<b>Chapter 3.0 AGILE SUMMARY .....</b>	<b>4</b>
Manifesto for Agile Development .....	4
Twelve Principles of Agile Software .....	5
What Agile is and isn't .....	6
The Chicken and the Egg.....	6
The Agile Sprint and Its Components.....	7
Daily Scrum.....	8
Burn Rate.....	9
Sprint Review Meeting.....	10
Sprint Retrospectives.....	10
Diagram of Major Sprint Components .....	11
<b>Chapter 4.0 USING SUITE WITH AGILE .....</b>	<b>12</b>
When to Use Agile.....	12
Agile Training.....	12
Agile vs. Waterfall Development .....	13
Agile Projects and the PMM.....	13
Project Tailoring .....	14
Agile Projects and the SEM.....	14
SEM Touch Points .....	14
Structured Walkthroughs .....	14
Stage Exits .....	15
Sprint Cycle .....	15
Sprint Review and Approval Template (SEM-0185) .....	16

**Privacy Information..... 17**

**Approval Information ..... 17**

    SUITE Express Integration for Agile Development..... 19

    Full SUITE Integration for Agile Projects..... 20

**REFERENCES..... 21**

**GLOSSARY..... 22**

**BIBLIOGRAPHY:..... 25**

## Chapter 1.0 INTRODUCTION AND BACKGROUND

The purpose of this guidebook is to provide background information, examples, and support for using Agile software development best practices in conjunction with the Department of Technology and Budget (DTMB) existing State Unified Information Technology Environment (SUITE) processes for Systems Engineering Methodology (SEM).

### *Why Agile, and Why Now?*

Agile best practices have been a part of mainstream software development for the past decade, and continue to evolve and gain acceptance. Agile adoptees expect to gain the following:

- More frequent, smaller product releases
- Shorter maintenance streams (inherent Agile Quality Assurance (QA) reduces post-production support)
- Quicker (more Agile) response to changing customer demands
- More consistent coding practices leading to uniform support across a development team (focus is on team-based support, not on a particular individual)

The call for Agile adoption has come from many sectors – large and small, public, private and government alike. Of particular interest to the DTMB, Agile best practices are equally well suited for application production support (enhancements, break-fix requests, etc.) and for new development efforts.

Like any other software development process, Agile best practices are no guarantee of success. The single best way to succeed has always been bringing the right people to support a project. DTMB has the right appropriate people, and Agile has an established record for improved software development. DTMB could have much to gain by moving forward with Agile adoption.

This process guide provides guidance on how to use Agile in conjunction with SEM, and also provides DTMB staff considering a move to Agile software development with important information.

As a contingency, if an Agile team is not ready to go forward, and must revert to pre-existing practices, all SEM documentation should remain current (in essence, Agile ‘Stories’ from which functional requirements are derived). The same SEM templates can continue to be used going forward with the original process – this underscores the importance of maintaining all SEM documentation with Agile development!

### *ITIL and CMMI*

Although beyond the scope of this guide, awareness of the Information Technology Infrastructure Library (ITIL) and the Capability Maturity Model Integrated (CMMI) is important, since these practices are also being introduced to DTMB business areas. In a

nutshell, both CMMI and ITIL focus on process improvement, with CMMI focusing on software development, while ITIL is broader-based including areas such as Infrastructure.

*“The basic difference between CMMI vs ITIL lies in application. While CMMI is focused toward software development, maintenance, and product integration, ITIL is broader in scope and provides a framework for IT service management and operations including a hardware life cycle.”*

The following four (4) web sites provide an excellent explanation of the purpose and overlap of the CMMI and ITIL models (more extensive documentation can be found via the internet):

Bright Hub: <http://www.brighthouse.com/office/project-management/articles/72298.aspx>

The Art of Service: <http://theartofservice.com/cmmi-vs-til-knowing-what-to-choose.html>

ITIL® Survival: <http://www.itil-survival.com/itilcmmi.html>

ITIL Official Site: <http://www.itil-officialsite.com/>

The key point to keep in mind is that CMMI, ITIL and Agile can (and often are) used together. For additional information on ITIL use, contact the Data Center Operations within DTMB.

**Chapter 2.0 SUITE SUMMARY**

The State Unified Information Technology Environment (SUITE) is an initiative to standardize methodologies, procedures, training, and tools for project management and systems development lifecycle management for all State of Michigan agencies. SUITE provides the framework for implementing repeatable processes in an effort to conduct systems development activities according to Capability Maturity Model Integrated (CMMI) Level 3 requirements. It is intended to ensure reliable, cost-effective, high quality Information Technology solutions that promote customer success.

CMMI is an industry standard for software maintenance and development. The focus of SUITE / CMMI is on the ability to manage the development, acquisition, and maintenance of technology products and services. CMMI is a model for process improvement that provides a common, integrated vision of improvement for all elements of a technology-based organization. It has been widely adopted by the private sector and is the most endorsed benchmark for delivery of reliable, high quality technology products and services. CMMI includes five (5) maturity levels: Initial, Managed, Defined, Quantitatively Managed and Optimizing. CMMI Level 3 (Defined) focuses on consistent, repeatable, managed processes across the entire technology organization.

SUITE primarily consists of the Project Management Methodology (PMM) and the Systems Engineering Methodology (SEM). The PMM provides standard methods and guidance to ensure that projects are conducted in a disciplined, well-managed and consistent manner. It promotes the delivery of quality products, and results in projects that are completed on time and within budget. The SEM provides guidance for information systems development related activities and software quality assurance practices. The primary purpose of the SEM is to promote the development of reliable, cost-effective, computer-based solutions while making efficient use of resources. By policy, all State of Michigan development projects must utilize the PMM and SEM.

**Chapter 3.0 AGILE SUMMARY***Manifesto for Agile Development*

In February 2001, a group of well-known software engineers met at Utah's Snowbird Ski Resort. They brainstormed ways to improve software development, based on their collective experience, and came up with the following manifesto:

*Manifesto for Agile Software Development*

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.

## Original Agile Manifesto Authors:

Kent Beck	Mike Beedle	Arie van Bennekum
Alistair Cockburn	Ward Cunningham	Martin Fowler
James Grenning	Jim Highsmith	Andrew Hunt
Ron Jeffries	Jon Kern	Brian Marick
Robert C. Martin	Steve Mellor	Ken Schwaber
Jeff Sutherland	Dave Thomas	

As a follow-up to the manifesto, they developed the following principles:

*Twelve Principles of Agile Software*

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.

### *What Agile is and isn't*

- Agile is a framework around which to build and continuously improve a team approach toward software development.
- Agile is often associated with a [non-exclusive] set of best practices, such as:
  - Paired programming
  - Swarming
  - Time Boxing
  - Scrum
  - Sprints (or iterations)
  - Test Driven Development (TDD)
  - Continuous Integration (CI)
  - Continuous Improvement (also CI!)

Note that many of these practices are used outside of the Agile world.

- [When done right] Agile IS structured!
- Agile is **not** a methodology. It is simply a four point manifesto and twelve guiding principles.
- Agile is NOT for everyone. It is not a cure-all or panacea; it is not right for everyone.

### *The Chicken and the Egg*

No Agile summary is complete without a reference to this commonly used fable:

A Pig and a Chicken are walking down the road. The Chicken says, "Hey Pig, I was thinking we should open a restaurant!" Pig replies, "Hmm, maybe, what would we call it?" The Chicken responds, "How about Ham and-Eggs?" The Pig thinks for a moment and says, "No thanks. I'd be committed, but you'd only be involved!" See the link:

[http://en.wikipedia.org/wiki/The\\_Chicken\\_and\\_the\\_Pig](http://en.wikipedia.org/wiki/The_Chicken_and_the_Pig)

The inference is that there are two kinds of project members: those who are committed to a project and accountable for its outcome, and those who just consult on the project and are informed of its progress.

### *The Agile Sprint and Its Components*

**Sprints** (also called iterations) are a core component of Agile. Briefly, a team defines a time period (typically 1 – 4 weeks) during which they will develop and deliver a working piece of software. Following are the major sprint elements:

- **A User Story** is a short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

*As a <type of user>, I want <some goal> so that <some reason>.*

User stories are often written on index cards or sticky notes, stored in a shoe box, and arranged on walls or tables to facilitate planning and discussion. As such, they strongly shift the focus from writing about features to discussing them. In fact, these discussions are more important than whatever text is written.

- **The Story or Product Backlog** is a prioritized features list, containing short descriptions of all functionality desired in the product. When using Scrum (an iterative and incremental Agile software development method for managing software projects and product or application development), it is not necessary to start a project with a lengthy, upfront effort to document all requirements. Typically, a Scrum team and its product owner begin by writing down everything they can think of easily. This is almost always more than enough for a first sprint. The product backlog is then allowed to grow and change as more is learned about the product and its customers.

A typical product backlog comprises the following different types of items:

1. Features
  2. Bugs
  3. Technical work
  4. Knowledge acquisition
- **In Scrum**, every iteration begins with a Sprint Planning Meeting. At this meeting, the Product Owner and the team review the outstanding stories and determine which ones will be tackled during that iteration/Sprint. Time-boxed to four hours, this meeting is a *conversation* between the Product Owner and the team. That is, it's up to the Product Owner to decide which stories are considered the highest priority to the release and which will generate the highest business value, but the team has the power to push back and voice concerns or impediments. This is a good thing since the team may be aware of a legitimate impediment keeping the team from moving forward.

When the team agrees to tackle the work, the Product Owner adds the corresponding stories into the Sprint Backlog. If a team uses manual Agile, this might be physically represented by moving a Post-It note or index card with a story written on it from the product backlog into the Sprint Backlog. If a team uses an Agile tooling solution, this might be represented in a number of ways. For instance, the tool ScrumWorks Pro® has a two-paned view of a project, in which the product backlog appears on the right and the sprint backlog on the left. Using an intuitive, drag-and-drop interface, the Product Owner can easily add stories to the sprint.

At this point, the Product Owner is typically asked to leave while the team decomposes the sprint backlog items into tasks. While the Product Owner is asked to leave so that the team can candidly discuss the work, he or she is still expected to be “on call” to answer questions, clarify acceptance criteria, or renegotiate. This meeting, which was previously called the sprint definition meeting, is also time-boxed to four hours to limit all sprint planning activities to a single day’s time.

- **The Sprint Backlog** is the list of work the team must address during the next sprint. The list is derived by selecting stories/features from the top of the product backlog until the team feels they have enough work to fill the sprint. This is done by the team asking “Can we also do this?” and adding stories/features to the sprint backlog. The team should keep in mind their previous sprints velocity (total story points from the last sprints stories) when selecting stories/features for the new sprint and use this number as a guideline of how much “effort” they can complete.

The stories/features are broken down into tasks by the team, which, as a best practice, should normally be between four and sixteen hours of work. With this level of detail the whole team understands exactly what to do, and potentially, anyone can pick a task from the list. Tasks on the sprint backlog are never assigned; rather, tasks are signed up for by the team members as needed during the daily scrum, according to the set priority and the team member skills. This promotes self-organization of the team, and developer buy-in.

The sprint backlog is the property of the team, and all included estimates are provided by the Team. Often an accompanying **Task Board** is used to see and change the state of the tasks of the current sprint, like “to do,” “in progress” and “done.”

- **Daily Scrum**  
Each day during the sprint, a project status meeting occurs. This is called a *daily scrum*, or *the daily standup*. This meeting has specific guidelines:
  - The meeting starts precisely on time
  - All are welcome, but normally only the core roles speak

- The meeting length is set to 15 minutes
- The meeting should happen at the same location and same time every day

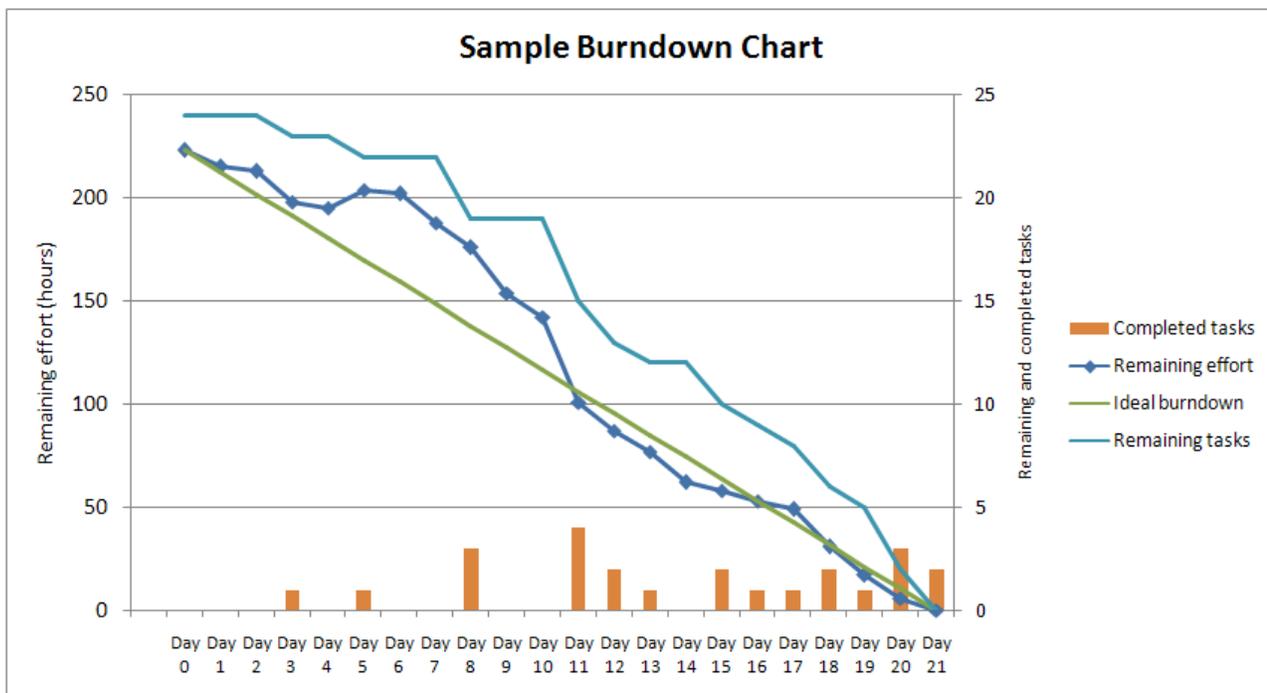
During the meeting, each team member answers three questions:

- What have you completed since yesterday?
- What are you planning to complete today?
- Any impediments/stumbling blocks?

It is the role of the ScrumMaster to facilitate resolution of these impediments, although the resolution should occur outside the Daily Scrum itself to keep it under 15 minutes.

● **Burn Rate**

The Burn Rate is viewed on a sprint Burn Down Chart showing remaining work in the sprint backlog. Updated every day, it gives a simple view of the sprint progress. It also provides quick visualizations for reference. There are also other types of burn down, for example the **release burn down chart** that shows the amount of work left to complete the target commitment for a Product Release (normally spanning through multiple iterations).



- **Sprint Review Meeting**

The purpose of the Sprint Review Meeting is to:

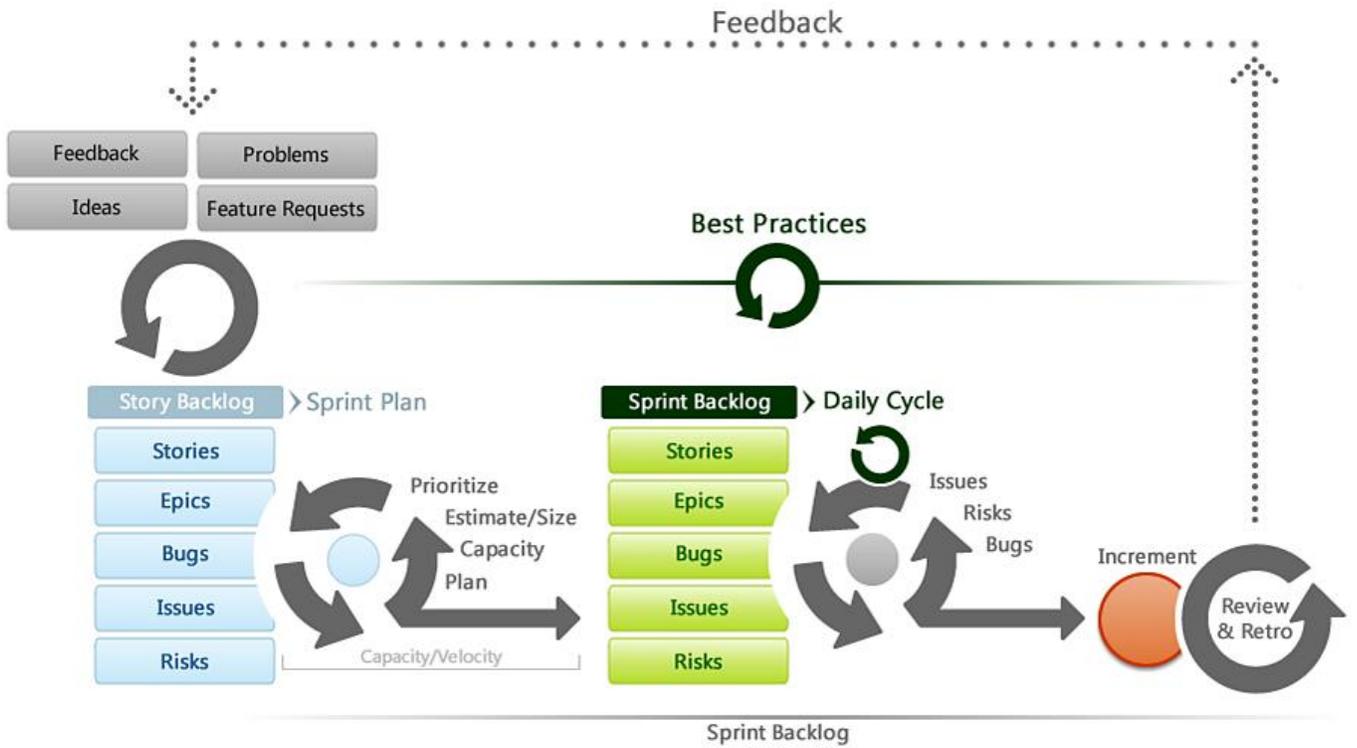
- Review the work that was completed and not completed
- Present the completed work to the stakeholders (a.k.a. “the demo”)
- Incomplete work cannot be demonstrated
- Four hour time limit

- **Sprint Retrospectives**

Sprint Retrospectives have the following characteristics:

- All team members reflect on the past sprint
- Team is tasked with making continuous process improvements
- Two main questions are asked in the sprint retrospective. What went well during the sprint? What could be improved in the next sprint? Many times, the team will prioritize items to be improved, and only select one or two, so that it can hold itself accountable for real progress at the next Retrospective.
- Three hour time limit

Diagram of Major Sprint Components



## Chapter 4.0 USING SUITE WITH AGILE

The objective of this chapter is to provide guidelines and instructions on applying SUITE when using Agile best practices, such as Scrum and Sprint iterations. To fully benefit, the users should have a good working knowledge of both SUITE and Agile best practices. For links and reference materials, see the References section of this document.

### *When to Use Agile*

Agile can be used for any project (for new development, as well as for break-fixes and application enhancements), but is much more likely to be successful when you have:

- The full support of management and the development team
- Customer support, since they are an active part of the team
- A project team that buys in to frequent, limited duration, face-to-face communication
- A stable, well defined source code repository and check-in process

When getting started with Agile, smaller projects are probably better than large ones as it gives the team a chance to get used to the new processes. Troubled projects, where team members are looking for a new approach to stalled development and delivery efforts, may actually be good Agile candidates as well. Conversely, using Agile will be much more challenging when:

- The project team is geographically dispersed (although there is technology available to help with this)
- The project team consists of more than 8 to 10 persons
- The customer is not readily available on a daily basis or simply doesn't see that as their role
- The project requirements are absolutely rigid and must be completely defined up front

### *Agile Training*

A well thought out Agile training plan (prior to using Agile!) is essential for everyone involved. It should be tailored to fit the team's needs, but usually includes the following:

- Definition of team member roles (explain 'Chicken and the Pig' metaphor)
- Scheduled visits to existing DTMB Agile Team Scrums ("You can't do Agile if you don't see Agile")
- Scheduled visits to existing DTMB Agile Team planning sessions
- Regular scheduled Agile best practice training sessions with these topics:
  - Scrum
  - Product Backlog
  - Sprint

- Demos
- Planning Sessions
- Team Retrospectives
- Continuous Integration
- Re-Factoring
- Test Driven Development
- Paired Programming and/or Swarming

If possible, it is a great advantage to provide Scrum Master Certification training for one person on the team. These sessions are usually two days, and cost less than \$1000.00. Many different vendors offer these certifications.

After training is complete, the team needs to decide which best practices (and what technology) to begin with, since doing them all at once is usually impossible. For example, you may begin using a white board and 3x5 note cards for your scrum sessions in the beginning, and once everyone gets used to the process, you can automate the process by using Microsoft Team Foundation Server (TFS) to track Sprint stories.

Once your Agile Sprints begin, you should expect the first few iterations to have less than perfect burn rates. It will take the team a few iterations to get used to the process, and it is important that the entire team makes improvements and adjustments accordingly at the retrospectives and planning sessions. **Failure to do this will almost certainly result in a failure of the Agile process!**

### ***Agile vs. Waterfall Development***

Agile is a different approach to system development than the traditional waterfall method. Under the waterfall approach, each SEM stage is completed and approved before the next stage can begin. For example, all Requirements are defined and approved prior to completing the Functional and System Design. They in turn are completed and approved prior to Construction and Testing. Under Agile, the system is defined, constructed and implemented over time, using a series of smaller efforts known as Sprints. In effect, each Sprint becomes an iteration of all the SEM stages, from Initiation and Planning through implementation.

### ***Agile Projects and the PMM***

For initiatives exceeding 200 hours, Agile development should be conducted within the confines of a normal project, using the State of Michigan's Project Management Methodology (PMM). Projects must be defined by a formal Project Charter, have a defined scope, and utilize all the usual project templates.

The only exception involves the Project Change Request (PMM-14). One of the principles of Agile development is that changing requirements are expected and welcome, even late in the project. Most changes will be adequately handled within the Sprint planning process. However, the Project Manager must review each requested change against the defined project scope. Changes that impact the scope of the project

should be documented using the Project Change Request (PMM-14).

Agile may be used with either the Full SEM or SEM Express. However, SEM Express is the recommended approach for most projects, as it is better suited for the iterative Sprint process and tends to require less overall documentation. Swim Lane diagrams depicting the use of Agile under each methodology are at the end of this section.

For projects less than 200 hours, the Systems Maintenance document (SEM-0931) can be used to capture all info related to the problem/enhancement. Story and task information can be entered on this, and related forms (SEM-402, 501, 604) as needed, with the main difference being that the number of people and Sprints required for completion would be smaller.

### ***Project Tailoring***

SUITE may be tailored to meet the business needs of Agile projects, just as it is permitted for non-Agile projects. The requested information must still be provided, in whatever manner makes the most sense. As an example, information related to functional requirements may be housed in TFS story and task descriptions, with a link to that information being provided on the SUITE form.

### ***Agile Projects and the SEM***

Agile projects must follow the State of Michigan's Systems Engineering Methodology (SEM) and produce all required SEM documentation. However, the various documents are not fully completed at the conclusion of each SEM stage. Rather, each SEM document is reviewed and updated as part of each Sprint. For example, once the content of a specific Sprint has been determined, the project team will update the Requirements Specification (SEM-0402), Functional Design Document (SEM-0501), System Design Document (SEM-0604), etc., as needed. At Project Closeout, each document should be complete and include the details of all the functionality provided by the project. Once created, supporting documents such as the Software Configuration Management Plan (SEM-0302) and Maintenance Plan (SEM-0301) may not need to be updated during a given Sprint, but should be reviewed during each and every Sprint.

### ***SEM Touch Points***

Project Managers must take care to properly utilize the SEM Touch Points and ensure deliverables dependent on other areas of DTMB are available when needed (i.e. Procurement, Infrastructure Services, Enterprise Security, etc.). Failure to do so could result in significant delays to individual sprints and the project as a whole.

### ***Structured Walkthroughs***

Structured walkthroughs are normally conducted for each major project deliverable. Under Agile, demos are provided to the project team at the conclusion of each Sprint or iteration. In effect, these demos are a series of structured walkthroughs and can be utilized in place of the standard process for each appropriate SEM deliverable. PMM deliverables should continue to be reviewed via the standard structured walkthrough

process.

### *Stage Exits*

Applicable SEM templates are created or updated (as needed) during each Sprint or iteration. All deliverables are listed on the Sprint Review and Approval form (SEM-0185) and the signatures on the form verify that the documents were reviewed and approved. In addition, a formal Stage Exit should be performed at the conclusion of each PMM Phase. At that time, all applicable PMM and SEM documentation should be reviewed for accuracy and completeness, using the Stage Exit Approval form (SEM-0189).

### *Sprint Cycle*

In each Sprint, the project team defines specific functionality to be coded, tested and implemented within a set timeframe (usually 1 to 4 weeks). As part of each Sprint, the project team performs the functional equivalent of the following:

1. Meet with end users to define the functionality to be included in the Sprint. This may include features from the Product Backlog that were not able to be completed during the previous Sprint, changes to previously implemented features, or completely new features or functionality. The agreed upon functionality must be attainable within the standard Sprint time frame that was established for the project.
2. Create a new Sprint Review and Approval form (SEM-0185) and complete the information in the Identification and Planning sections. Formal approval is obtained from the Project Sponsors for the content and planned implementation date of the Sprint. *See Exhibit 2.0-3, Sprint Review and Approval Form.* (This process can also be tailored with an equivalent process. For example, the customer may use TFS task status to essentially ‘sign off’ on the functionality. This should be understood and agreed upon by all.)
3. Using the SEM, develop the Requirement Specification, Functional Design, System Design, Test Plans, etc. for the functionality included in the Sprint. In the first Sprint, each SEM document will need to be created. For each subsequent Sprint, the SEM document will be updated accordingly. In actual practice, a hybrid approach may occur, where foundational infrastructure and design work (amounting to 10-20% of the project) is defined prior to beginning the first Sprint.
4. Develop functionality and complete testing for backlog items included in the Sprint.
5. Perform all necessary reviews and Structured Walkthroughs and ensure all pertinent documentation has been updated.
6. Update the Implementation Date in the Sprint Review and Approval form (SEM-0185) and obtain formal approval for implementation from the Project Sponsors.

7. Implement the added functionality included in the Sprint. Depending on how your team approaches implementation, this can consist of pushing the new functionality to either the Quality Assurance (QA) or Production environment.
8. Any planned functionality not implemented during the Sprint is added to the Product Backlog for consideration in subsequent Sprints.

### ***Sprint Review and Approval Template (SEM-0185)***

The Sprint Review and Approval Form (SEM-0185) is a new SEM form for documenting the planning and implementation of each new Sprint. A separate form is required for each Sprint performed during the life of the project.

The Sprint Review and Approval Form provides two important functions. First, it documents the planned functionality to be implemented by the Sprint and the planned implementation date. Secondly, it provides formal approval for the new functionality to be implemented. That approval also includes certification that all Structured Walkthroughs were performed and all pertinent documentation has been properly updated.

The specific fields on the Sprint Review and Approval Template include:

1. System or Project Name
2. Sprint Number / Description
3. Deliverables Subject to Approval
4. Scrum Master Name
5. Metrics:
  - a. Sprint Planning Begin Date
  - b. Planned Sprint Implementation Date
  - c. Actual Sprint Implementation Date
6. Sprint Overview (narrative of the included features and functionality)
7. Sprint Planning Approval (Name / Title, Signature, and Date):
  - a. Client Agency Project Sponsor or Representative
  - b. DTMB Project Sponsor or Representative
  - c. Scrum Master
8. Sprint Implementation Approval (Name / Title, Signature, and Date):
  - a. Client Agency Project Sponsor or Representative
  - b. DTMB Project Sponsor or Representative
  - c. Scrum Master

**Sprint Review and Approval Form (SEM-0185)**

**State of Michigan**  
**(Insert System or Project Name Here)**  
**Sprint Review and Approval**

*Project Information*

1. System or Project Name		
2. Sprint Number/Description		
3. Deliverables Subject to Approval		
4. Scrum Master Name		
5. Metrics		
a. Sprint Planning Begin Date	b. Planned Sprint Implementation Date	c. Actual Sprint Implementation Date

*Approval Information*

**Sprint Overview:**

**Sprint Planning Approval Signatures**

Role	Name/Title	Signature	Date
Client Agency Representative			
DTMB Representative			
Scrum Master			

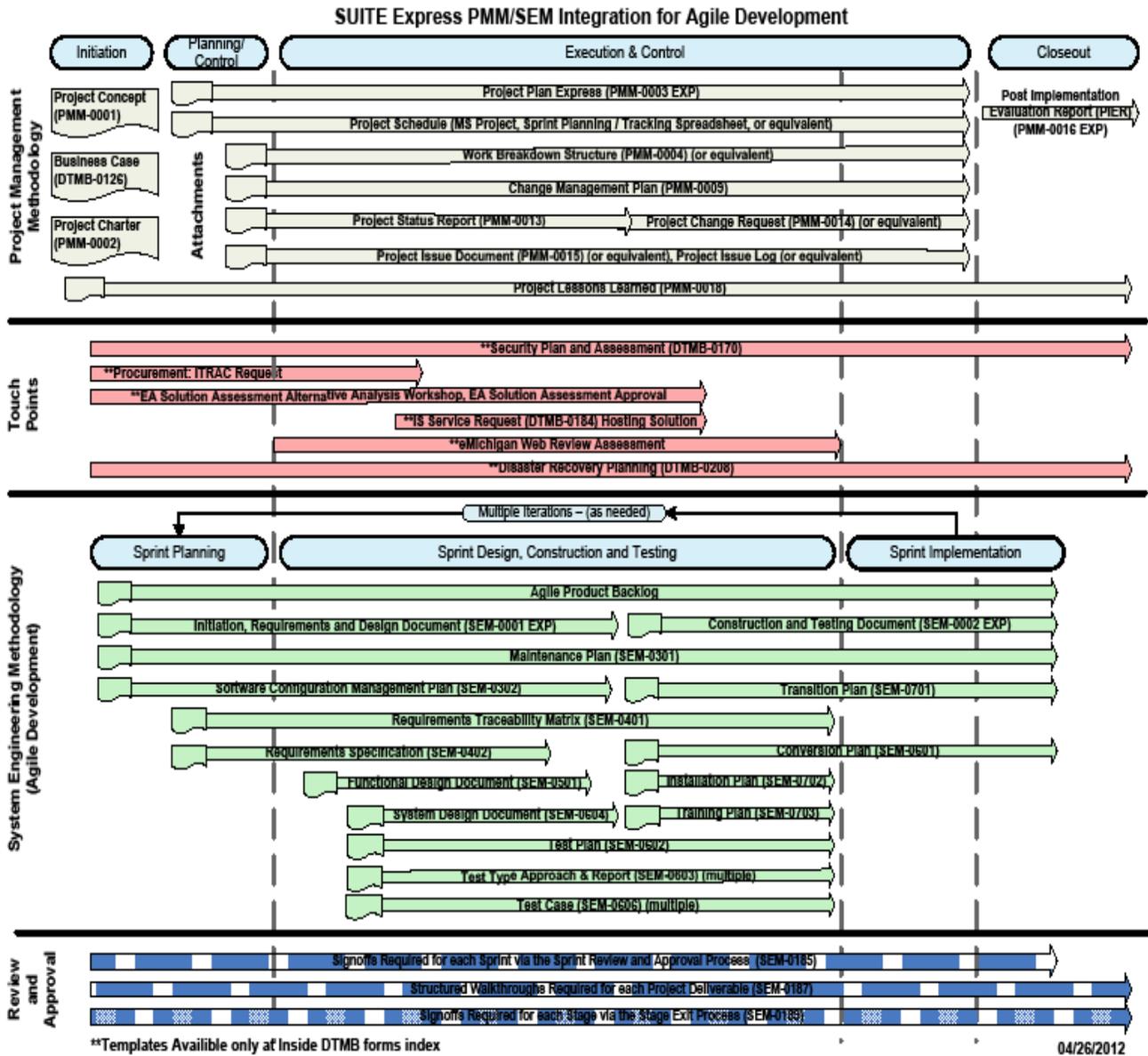
**Sprint Review and Approval Form (SEM-0185) (continued)**

**Sprint Implementation Approval Signatures**

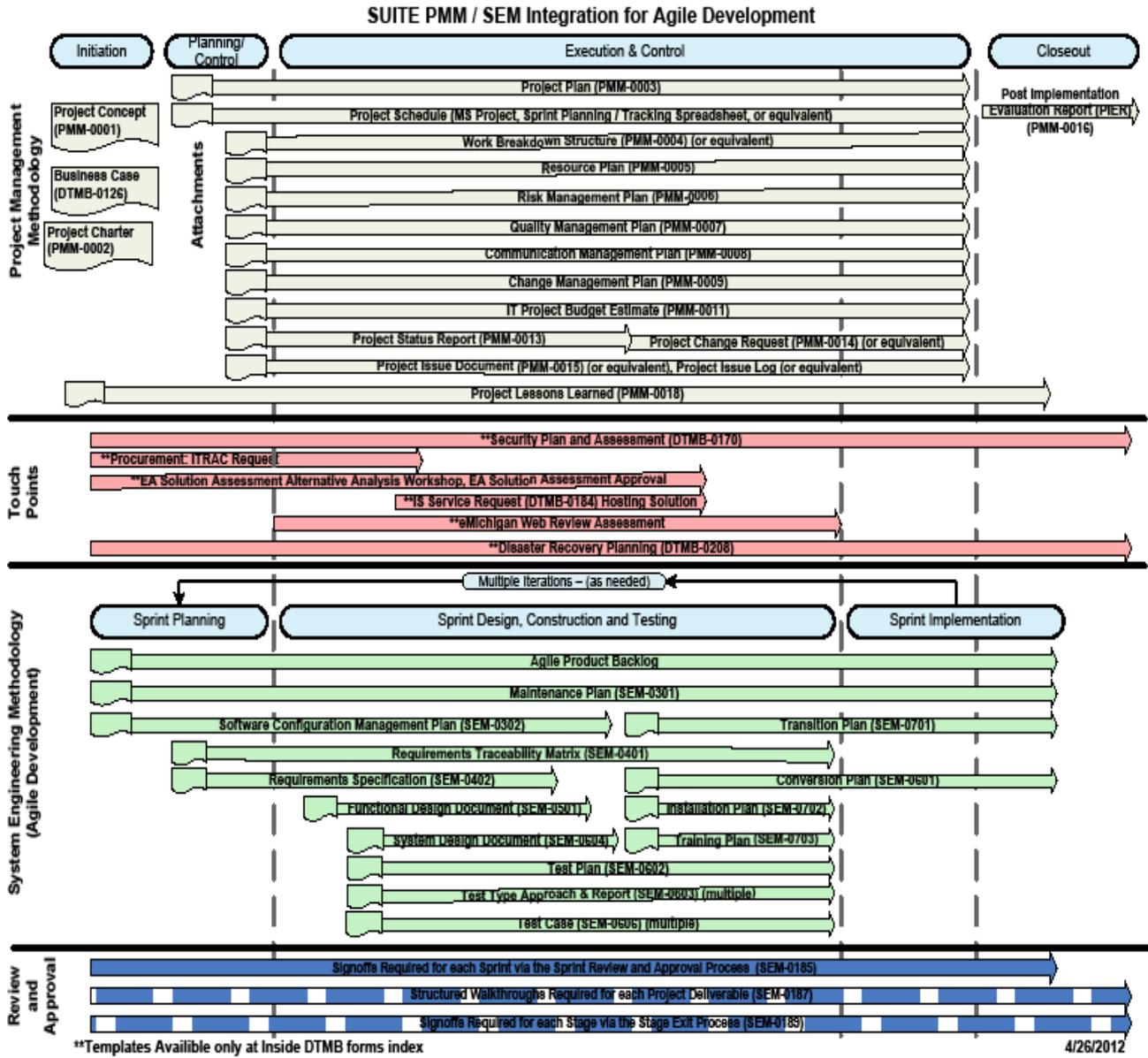
<b>Role</b>	<b>Name/Title</b>	<b>Signature</b>	<b>Date</b>
Client Agency Representative			
DTMB Representative			
Scrum Master			

**Sprint Review and Approval Form (SEM-0185) (continued)**

SUITE Express Integration for Agile Development



Full SUITE Integration for Agile Projects



**REFERENCES**

1. DTMB Agile PAT SharePoint site:  
<http://inside.michigan.gov/sites/dtmb/epmo-pmo/suite/Agile%20PAT/Forms/AllItems.aspx>
2. Manifesto for Agile Software Development  
<http://www.Agilemanifesto.org/>
3. Federal Agile Blog  
<http://www.federalAgilesolutions.com/>
4. 25 Point Plan to Reform Federal Information Technology Management  
<http://www.cio.gov/documents/25-Point-Implementation-Plan-to-Reform-Federal%20IT.pdf>
5. Wikipedia Scrum (Development)  
[http://en.wikipedia.org/wiki/Scrum\\_\(development\)](http://en.wikipedia.org/wiki/Scrum_(development))
6. Martin Fowler – Continuous Integration  
<http://www.martinfowler.com/articles/continuousIntegration.html>
7. Wikipedia – Pair Programming  
[http://en.Wikipedia.org/wiki/Pair\\_programming](http://en.Wikipedia.org/wiki/Pair_programming)
8. Wikipedia – Agile Software Development  
[http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development)
9. Find White Papers – An Introduction to Agile Software Development  
<http://www.findwhitepapers.com/content11433>
10. TechLife Columbus – The Promise and Reality of Agile after 10 Years  
<http://www.meetup.com/techlifecolumbus/events/28186891/?eventId=28186891&action=detail>
11. Broadsword  
<http://www.broadswordolutions.com/resources/>
12. Top 5 Software Development Process Challenges (white paper)  
[http://informationweek.com/whitepaper/development/tools/top-5-software-development-process-challenges-wp1318601421?articleID=191703607&cid=wallstreetandtech\\_fture\\_wp\\_default&itc=wallstreetandtech\\_fture\\_wp\\_default](http://informationweek.com/whitepaper/development/tools/top-5-software-development-process-challenges-wp1318601421?articleID=191703607&cid=wallstreetandtech_fture_wp_default&itc=wallstreetandtech_fture_wp_default)
13. ScrumWorks Pro®  
<http://www.open.collab.net/products/scrumworks/capabilities.html>
14. Pro-Tech Professional Technical Services, Inc. – Scrum Cheat Sheet  
<http://www.protechtraining.com/pdf/ScrumCheatSheet.pdf>

---

**GLOSSARY**

**Acceptance Criteria** – The criteria that have to be met for a story to be assessed as complete. Link: <http://scrummethodology.com/scrum-acceptance-criteria/>

**Backlog** – A comprehensive to-do list, expressed in priority order based on the business value each piece of work will generate. Link: <http://scrummethodology.com/the-scrum-backlog/>

**Baseline** – An agreed-to description of the attributes of a product, at a point in time, which serves as a basis for defining change. (2) An approved and released document, or a set of documents, each of a specific revision; the purpose of which is to provide a defined basis for managing change. (3) The currently approved and released configuration documentation. (4) A released set of files comprising a software version and associated configuration documentation.

**Burn Down Chart** – A graphical representation of the work left to do versus time. Link: [http://en.wikipedia.org/wiki/Burn\\_down\\_chart](http://en.wikipedia.org/wiki/Burn_down_chart)

**Burn Rate** – The rate at which hours allocated to a project are being used. Link: [http://en.wikipedia.org/wiki/Burn\\_rate](http://en.wikipedia.org/wiki/Burn_rate)

**Configuration Audit** – Configuration Audits determine to what extent the as-designed/as-tested/as-built product reflects the required physical and functional characteristics specified in the requirements. Functional Configuration Audit (FCA) and Physical Configuration Audit (PCA) are done once, to establish the Product Baseline.

**Development Baseline** – The baseline comprising the software and associated technical documentation that define the evolving configuration of the system during development. This baseline includes the software design, and implemented code, database schema, and COTS products, and evolves into the product baseline.

**Fix number** – An indicator of small updates that are to be built into a regular modification or release at a later time. The version, release, modification, and fix levels together comprise the program level (or version) of a program.

**Functional Baseline** – The baseline comprising documentation and possibly models that specify system functional, data, and technical requirements.

**Functional Configuration Audit (FCA)** – An inspection to determine whether the (software) configuration item satisfies the functions defined in specifications. Consists of someone acknowledging having inspected or listed each item to determine it satisfies the functions defined in specifications.

**Impediment** – In Scrum, an impediment is anything that keeps a team from being productive.

Link: <http://scrummethodology.com/scrum-impediments/>

**Iteration** – A distinct sequence of activities with a baselined plan and valuation criteria resulting in a release.

**Modification number** – The modification level of a program which is an indicator of changes that do not affect the external interface of the program. The version, release, modification, and fix levels together comprise the program level (version) of a program.

**Physical Configuration Audit (PCA)** – The formal examination of the "as-built" configuration of a configuration item against its technical documentation to establish or verify the configuration item's product baseline.

**Product Owner** – The primary person responsible for a project's success. The Product Owner leads the development effort by conveying his or her vision to the team, outlining work in the scrum backlog, and prioritizing it based on business value. Link: <http://scrummethodology.com/scrum-product-owner/>

**Program level** – The version, release, modification, and fix levels of a program.

**Regression Testing** – The process of running a composite of comprehensive test cases that are always run after system modifications to detect faults introduced by modification.

**Release number** – An indicator of changes to the external programming interface of the program. The version, release, modification, and fix levels together comprise the program level (version) of a program.

**Scrum** – An iterative and incremental Agile software development method for managing software projects and product or application development.

Link: [http://en.wikipedia.org/wiki/Scrum\\_\(development\)](http://en.wikipedia.org/wiki/Scrum_(development))

**Scrum Master** – Team member who ensures the process is followed, removed impediments, and protects the development team from disruption.

**Specification** – A document that explicitly states essential technical attributes/requirements for a product and procedures to determine that the product's performance meets its requirements/attributes.

**Version** – (1) One of several sequentially created configurations of a data/document product. (2) A supplementary identifier used to distinguish a changed body or set of computer-based data (software) from the previous configuration with the same primary identifier. Version identifiers are usually associated with data (such as files, databases and software) used by, or maintained in, computers.

**Version Description Document (VDD)** – A document associated with a product release that describes the version released and describes the versions of the items included in the product.

**Version Number** – An indicator of the hardware and basic operating system upon which the program operates. The version, release, modification, and fix levels together comprise the program level (version) of a program.

**BIBLIOGRAPHY:**

The following materials and web sites were referenced in the original preparation of this guide.

1. State of Michigan Project Management Methodology:  
<http://www.michigan.gov/suite>
2. Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York, NY: 1990.
3. The Institute of Electrical and Electronics Engineers, Inc., *IEEE Standard for Developing Software Lifecycle Processes*, IEEE Std 1074-1991, New York, 1992.