First Tech federal credit union

SDLC

Software Development Life Cycle

First Tech Federal Credit Union

Draft Version

Version History

Version	Date	Prepared By	Changes	Comments
1	3/12/2018	Rajesh Sivakumar	Initial Draft	
2	5/24/18	Grant Gaines	Non dev areas	Worked with teams to fill in non dev sections such as security.

Table of Contents

Overview

To keep up with the fast-paced change of Member needs and to produce software with the highest quality and lowest cost in the shortest time, First Tech Federal Credit Union (FTFCU) has adopted the Agile Software development methodology, which offers an iterative approach to design and development of software. The Agile approach embraces the constant changes that occur in the development of technology - allowing teams to break the lengthy requirements, build, and test phases down into smaller segments, ultimately delivering working software quickly and more frequently.

One major benefit of using the Agile approach is to reduce risk to the project by validating requirements early in the software development life cycle. Risk is also reduced by introducing “spikes”. A spike is a short time-boxed effort to explore an approach, investigate an issue or reduce a project risk. It is used to investigate any aspect of the project including technical, business and interpersonal issues. If the team’s early proof of concept efforts is unable to eliminate a critical risk or technical challenge, it indicates that the project is not viable and allows resources to be assigned to other projects/approaches

It may not be possible to apply this methodology to all the projects; however enough effort must be put in to adapt the agile methodology and exceptions need to be approved.

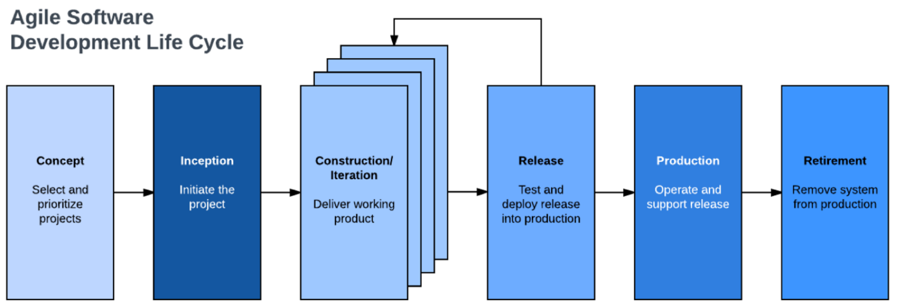
If the project is Vendor focused, we will need a copy of the vendor’s SDLC and will need to agree on Compliance. This will need to be done as part of the VMO process.

This document represents the process that will be followed by all departments that interact with the SDLC. This document is considered a living document and will be reviewed periodically to ensure FTFCU is meeting the industry best practices. This document is owned by Enterprise Project Management Office and is approved by the Board of Directors annually. All software development projects and programs are required to follow this life-cycle.

In order to facilitate consistent quality deliverables, departments within FTFCU must strive and work toward a repeatable and sustainable process. This document/process represents the framework that FTFCU follows when providing quality deliverables. Incorporating industry best practices requires coordination between various departments. Direction among departments ensures that FTFCU delivers a high-quality product that meets or exceeds the acceptance criteria:

FTFCU Agile Process Flow

The following view presents the full Agile lifecycle model within FTFCU. There may be projects operating simultaneously, multiple sprints/iterations being logged on different product lines, and a variety of business units, with a range of business needs.


Agile Software Development Life Cycle
Concept
Select and prioritize projects
Inception
Initiate the project
Construction/ Iteration
Deliver working product
Release
Test and deploy release into production
Production
Operate and support release
Retirement
Remove system from production

- Concept

  Projects are envisioned and prioritized.

  During the first step of the software development life cycle, the team scopes out and prioritizes projects. Some teams may work on more than one project at the same time depending on the department's organization.

  For each concept, we should define the business opportunity and determine the time and work it'll take to complete the project. Based on this information, we can assess technical and economic feasibility and decide which projects are worth pursuing.

- Inception

  Team members are identified, funding is put in place, and initial environments and requirements are discussed.

  Once you have identified the project, work with stakeholders to determine requirements. You might want to use user flow diagrams or high-level UML diagrams to demonstrate how the new feature should function and how it will fit into your existing system. The output of this phase will be a Solution Design Document that has to be reviewed and approved by a committee. The Solution Design Document should be a collaborative effort by personnel from the various IT departments like Cyber Security, Infrastructure, Application Development, CBA…

Once the Solution Design is approved, team members to work on the project are selected and allocate resources. Create a timeline or a swim lane process map to delineate responsibilities and clearly show when certain work needs to be completed for the duration of the sprint.

- Iteration/Construction

  The UX, Development, QA, Infrastructure, InfoSec, CBA teams works to deliver working software based on iteration requirements and feedback.

  Once a team has defined requirements for the initial sprint based on stakeholder feedback and requirements, the work begins. UX designers and developers begin work on their first iteration of the project, with the goal of having a working product to launch at the end of the sprint. Remember, the product will undergo various rounds of revisions, so this first iteration might only include the bare minimum functionality. The team can and will have additional sprints to expand upon the overall product.

  Here are the various phases of the development cycle:
  - **Requirements Gathering**
  - **Detail Design**
  - **UX Design**
  - **Design Review** (pull in other IT team members as needed)
  - **Prototyping**
  - **Test Case Construction**
  - **Test Case Review** (pull in other IT team members as needed)
  - **Load Testing Plan Creation/Review** (may be part of the last couple of sprints)
  - **Development**
  - **Unit Testing**
  - **Dev Testing**
  - **Code Review**
  - **Cyber Security Review** (may be part of the last couple of sprints)
  - **UX Review**
  - **QA Testing**
  - **UAT**

- Release

  QA (Quality Assurance) testing, Security Testing, internal and external training, documentation development, Load Testing, and final release of the iteration into production.

  You’re nearly ready to release your product to production. Finish up this software iteration with the following steps:

  - **Test the system.** Your quality assurance (QA) team should test functionality, detect bugs, and record wins and losses.
  - **Address any defects.**
  - **Load Testing** (done as part of the last couple of sprints)

- **Finalize system and user documentation.**
- **Formalize post production Support plan**
- **Perform the necessary Security scans and remediate any findings.**
- **Release the iteration into production.**

- Production

  Ongoing support of the software.

  This phase involves ongoing support for the software release. In other words, your team should keep the system running smoothly and show users how to use it. The production phase ends when support has ended or when the release is planned for retirement.

- Retirement

  End-of-life activities, including customer notification and migration.

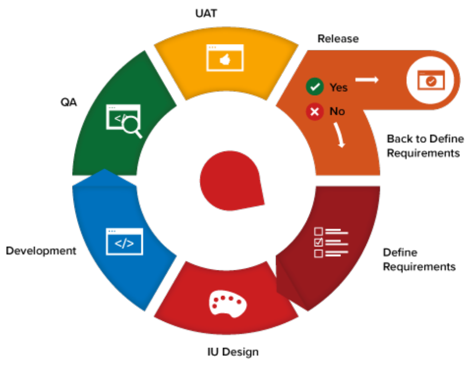
  During the retirement phase, you remove the system release from production, typically when you want to replace a system with a new release or when the system becomes redundant, obsolete, or contrary to your business model.

Agile Iteration Workflow

Each iteration is usually multiple weeks in length and has a fixed completion time. Multiple iterations will take place during the lifetime of the project. During an iteration, it is important that the business stakeholders provide feedback to ensure that the feature meets their needs.

A typical iteration process flow is as follows:

- **Requirements** - Define the requirements for the iteration based on the product backlog, sprint backlog, customer and stakeholder feedback.
- **Development** – Design and Develop software based on defined requirements
- **Testing** – QA testing and documentation in a pre-production environment
- **Delivery** – Integrate and deliver the working iteration into production
- **Feedback** – Accept customer and stakeholder feedback and work it into the requirements of the next iteration



Sprint planning

Work is divided into sprints, with the goal of producing a working product at the end of each sprint. A sprint typically lasts two or three weeks. The workflow of a sprint should follow this basic outline:

- **Plan.** The sprint begins with a sprint planning meeting, where team members come together to lay out components for the upcoming round of work. The product manager prioritizes work from a backlog of tasks to assign the team.
- **Develop.** Design and develop the product in accordance with the approved guidelines.
- **Test/QA.** Complete thorough testing and documentation of results before delivery.
- **Deliver.** Present the working product or software to stakeholders and customers.
- **Assess.** Solicit feedback from the customer and stakeholders and gather information to incorporate into the next sprint.

In addition to sprint planning meetings, your team should gather for daily meetings to check in and touch base on the progress, hash out any conflicts, and work to keep the process moving forward.

Commented [DLS1]: You might consider mentioning that an Agile "Sprint" is similar to a Milestone in traditional waterfall project planning. That will help understanding for non-project management professionals.

Commented [DLS2]: Project plans will have specific dates for deliverables but the SDLC should be more general.

Sprint Work Item Lifecycle

- Work items flow from the backlog in an **To Do/Open** status to the **Closed** status
- Task States
    - **To Do**

      Represents all user stories and defects in the backlog that are not currently started
    - **In Progress**

      Developer/UX Designer/QA has currently started working on the item. If the item is a development item, it is typically started with the team member creating a feature branch in their forked repository. If a design review is necessary, the developer will either have an architect and/or UI designer review both the application design as well as the UI design as appropriate. Before the item can leave In Progress, the developer must have completed the following:

        - Create appropriate unit tests
        - Validated that the work item resolves all of the success criteria
        - Verify tests pass locally
        - Changes checked in change set into their working repository
        - A pull request is created in GitHub Enterprise to merge the changes into the build branch

      Once development is complete, the work item will either go into Code review. All work items must be code reviewed by a peer.
    - **Code Review**

      A peer is responsible for doing a thorough code review. The review should look for the following:

        - SQL Injection
        - Cross-site scripting exploits
        - Inappropriate storing of confidential information
        - Homegrown encryption or hashing
        - Logging according to our best practices document
        - Checking the result of service calls in client repositories and other areas to correctly handle error conditions resulting from a WCF service call
        - Migrations written according to our best practices document and documented in our developer notes spreadsheet.

      The reviewer will use GitHub Enterprise to review the changes and document any issues that were found. If the peer finds issues with the code and rejects the change set, the work item will move back into In Progress. If the lead accepts the change set, the work item will move into QA Validation.

    - **QA Validation**

      QA validates that the work items' acceptance criteria is valid. QA will begin their testing. If the work item fails the tests, the work item is moved back into the In-Progress state for the developer to resolve. If the work item successfully passes the tests, the work item is Resolved.

- **Resolved**
  A work item is considered Resolved when the work item is checked in, successfully builds and all tests pass on the CI server. The feature has been reviewed. The acceptance criteria have been met. Code and implementation has been reviewed. QA defined tests have been executed and pass. All relevant integration tests have been created and pass on the CI server. The new feature has been deployed to QA environment and verified. If the work item fails, the work item is Reopened and placed back in an Open state for a developer to resolve.

- **Closed**

  A work item is considered Closed when it has been resolved and successfully tested in the Test/Staging environment. If the work item fails, the work item is Reopened and placed back in an In-Progress state for a developer to resolve.

Continuous Integration

Continuous Integration (CI) is a process and setup of automated tools that are intended to provide rapid feedback to developers about the current state of the code-base. The main goal is to fail fast and fail early, so that corrections can be made as early in the process as possible.

The CI server will be configured to start builds as code is checked into source control. In addition to build jobs, the server will also support additional test jobs, release packaging jobs and deployment jobs.

Build Job Tasks

- Poll source control for changes
- Pull down code from appropriate branch in GitHub Enterprise
- Pull in required dependencies from the binary repository
- Compile code
- Run unit tests
- Run additional code coverage, complexity and static analysis tools as required
- Package and archive build output to the binary repository
- Publish results and other build metadata

Deployment Job Tasks

- Identify the release to deploy and the relevant target environment
- Pull the release from the binary repository and stage for installation
- Install the release in the target environment
- Perform environment specific configuration changes
- Perform any required installation validation
- Perform automated and manual smoke tests to validate the release.

Continuous Delivery

Continuous Delivery is an extension of Continuous Integration. It is the processes and tools that allow release packaging and deployment to happen automatically and often. Like the CI system, it should be designed to fail fast and fail early. This provides faster feedback when there is a problem.

Continuous delivery does not necessarily mean that every release automatically is deployed to production. Business requirements may make that undesirable. Instead, releases can be delivered continuously to internal test and staging environments.

Continuous delivery process is integrated into the jobs running on the CI server. Primarily the Release Packaging and Deployment jobs, also additional Test jobs needed to validate the entire release package. Once a release has been created the following activities may be performed. This is a partial list of possible post-release activities. Many of these can be run in parallel, with some requiring specific test outcomes before executing.

Post Release Processes

- High-level system functional tests
- High-level UI based automated tests
- System load testing
- System performance testing
- System security testing
- Configuration based testing
- Internal production deployments
- External production deployments.

Assembly and Release Versioning

Component and Release versioning will follow the Semantic Versioning 2.0.0 guidelines, available here: semver.org. This versioning scheme will be applied to components when they are built and to releases when they are packaged.

Release Packaging

Releases will be versioned as part of the packaging process.

Deployment

Deployments need to be automated and reproducible. Environment specific configuration details will be externalized so that deployment scripts can easily adjust depending on the targeted environment.

Deployment of a new release to an environment needs to ensure that all of the release contents are installed correctly and there aren’t any old artifacts left behind from previous releases. Deployment scripts should perform install validation to be considered successful.

Hotfixes and Patching

A hotfix requires a starting point that matches the release that is being patched. Release metadata will preserve the appropriate change sets so that a hotfix branch can be created correctly. It is only necessary to branch the solutions involved in the hot fix.

These branched solutions will be built on the CI server in the same manner as regular builds. Once the fix is complete and the tests pass, it can be packaged into a new release. The new release will use the same versions of the components from the original release, but with updated versions of the components changed to addresses the bug. This new release can then be tested and deployed like any other release.

If the hotfix changes are relevant to the mainline code, they can be merged back into mainline. It is not necessary to automatically merge the changes back into mainline.

Quality Assurance (QA)

Quality Assurance (QA) activities are conducted in every phase of the SDLC process. Throughout the development process, QA is actively writing test cases, test scripts, clarifying requirements, identifying the acceptance test criteria, gathering test data, and ensuring the QA test environment is as comparable to the production user experience as possible.

Security

Security needs to be incorporated into the Secure SDLC process to ensure that security assurance activities such as penetration testing, code review, and architecture analysis are a part of the development effort. Utilizing a Secure SDLC approach ensures:

- More secure software as security is a continuous concern
- Awareness of security considerations by stakeholders
- Early detection of flaws in the system
- Cost reduction as a result of early detection and resolution of issues
- Overall reduction of intrinsic business risks for the organization

Further requirements including security review of development efforts please reference the Secure Software Development Review Policy located on the Cyber Security HUB page.

Project Delivery (PMO)

Change Management

- First Tech has an established Change Management Policy (CMP) for configuration and scope changes associated with production environments.
- Security scan results must be uploaded to Change Management system before Cyber Security approval can be given
- Have a corresponding Request for Change (RFC) completed
- Have business risk identified
- Have a installation start and end date and time defined
- Have supporting documentation per the Business Risk Matrix
- Have test results attached (when possible)
- Be performed in approved maintenance windows (when possible)
- Be approved by the Business Unit
- Be approved by the RFC Owner
- Have segregation of duties between the Requestor, Owner and Implementer
- Be performed by authorized personnel
- Have communication sent to stakeholders with a minimum of one business day notice for any impacting change (when possible)
- Be tracked, measured and reported upon

Software Release Management / Installation

Software releases are managed as part of the Continuous Delivery (CD) process described above. Software releases are delivered to the implementation team via the CD tool for deployment to the production environment. RBAC has been implemented in the CD tool so only authorized personnel are allowed to deploy software to the production environment. Any changes to databases are performed via scripts provided by the development team. Which software versions are installed into the production environment are available via a dashboard in the CD tool.

Application Support

Once the application has been deployed to the production environment primary support duties are turned over to the operations team. With assistance from the development team support documentation is developed and stored on the IT wiki. The following topics are covered in the documentation.

- Environment architecture diagrams
- Support plan
- Support contacts
- Support processes
- Downtime procedures
- Smoke test instructions
- Troubleshooting information
- Training videos/documentation
- Monitoring procedures
- Batch job information
- Document cold storage
- Disaster recovery
- Process dataflow diagrams
- Software installation instructions

Issues with the application in the production environment are investigated by the operations team. When needed the operations team will pull in resources from across IT to resolve the issue. Regular maintenance and support activities are carried out by the operations team as needed.

Disaster Recovery

Disaster Recovery must be built into every software development project.

Starting a new product SDLC:

- Identify the likely recovery criticality and priority (BIA) once this is in production.
- Identify backup, imaging, or replication solutions that meet likely RTO and RPO.
- Integrate the DR implementation steps into the Agile project plan.
- Incorporate a recovery or failover exercise in the project plan prior to release.
- Add this product to the DR Life-Cycle project plan (link).

Existing products entering a new SDLC:

- Review and update the BIA (business impact analysis) .
- Review how the SDLC changes might impact existing recovery solutions, RTO, & RPO.
- Investigate opportunities to improve existing recovery strategy and methodology.
- Integrate any DR changes into the Agile project plan.
- Incorporate a recovery or failover exercise in the project plan prior to release.
- Make adjustments to the DR life-cycle project plan if needed.

Coding Style Guide and Standards

Microsoft Azure

Azure Security Best Practices and Patterns
https://docs.microsoft.com/en-us/azure/security/security-best-practices-and-patterns

Azure Architecture Center
https://docs.microsoft.com/en-us/azure/architecture/

Batch Services

Perl Coding Style Guide
https://perldoc.perl.org/perlstyle.html

cView

SQL Standards and Conventions
http://www.sql-server-performance.com/2001/sql-best-practices/
http://www.itprotoday.com/software-development/t-sql-best-practices-part-1

Salesforce Standards and Conventions (FirstApp, FirstTouch)

Apex Best Practices
https://developer.salesforce.com/page/Apex_Code_Best_Practices

Salesforce Best Practices
https://developer.salesforce.com/docs/atlas.en-us.salesforce1.meta/salesforce1/dev_best_practices.htm

https://developer.salesforce.com/blogs/engineering/2015/05/developer-practices-checklist.html

https://developer.salesforce.com/docs/atlas.en-us.salesforce1.meta/salesforce1/salesforce1_guide_introduction.htm

Groovy Coding Style

http://groovy-lang.org/style-guide.html

OLB/OMB/BizTalk/API

C# Coding Conventions

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions/

Secure Coding Guidelines

https://docs.microsoft.com/en-us/dotnet/standard/security/secure-coding-guidelines/

Asp.Net MVC Best Practices

https://blogs.msdn.microsoft.com/aspnetue/2010/09/17/best-practices-for-asp-net-mvc/

MobileX

React/ReactJs Coding Guidelines
Airbnb React/JSX Style Guide

https://github.com/airbnb/javascript/tree/master/react/

https://medium.freecodecamp.org/10-points-to-remember-thatll-help-you-master-coding-in-reactjs-library-d0520d8c73d8/

React/React-Native Code Review Checklist

https://ftagile.atlassian.net/wiki/spaces/AR/pages/594477100/React+React-Native+Code+Review+Checklist

Enterprise Data Warehouse (EDW)

ETL Standards

https://ftagile.atlassian.net/wiki/spaces/DI/pages/388595831/ETL+Standards

EDW Design Principles

https://ftagile.atlassian.net/wiki/spaces/PAG/pages/91717982/EDW+Design+Principles

Enterprise Data governance

https://ftagile.atlassian.net/wiki/spaces/PAG/pages/90898508/Enterprise+Data+Governance+EDG

EDW: Service level agreements

https://ftagile.atlassian.net/wiki/spaces/PAG/pages/108659377/EDW+Service+Level+Agreements

Access & Security
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/91717752/Access+Security

EDW Data refresh Schedule
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/91717750/EDW+Data+Refresh+Schedule

EDW PII Attributes
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/119186844/EDW+PII+Attributes

Data Quality
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/91718426/Data+Quality

Data classification
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/91718419/Data+Classification

Data security
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/91718429/Data+Security

Tableau
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/15990963/Tableau

Alteryx Desktop /Server requirements
https://ftagile.atlassian.net/wiki/spaces/PAG/pages/21561534/Alteryx+DeskTop+Server+Requirements

QA Automation Tests

Java Coding Style guide
https://google.github.io/styleguide/javaguide.html