Software Lifecycle Models

James Falkner, Sun Microsystems, Inc.

October 31, 2005

Table of Contents:

I. Introduction

Describing the various software lifecycles at a high level helps to provide a basis for the identification of the overall phases that corporations, its partners, and its customers experience when producing, delivering, deploying and maintaining software systems using SDD standards.

Understanding the phases in each lifecycle helps one to understand the participants in the overall lifecycle. Participants may be translated into stakeholders in the architecture, user profiles and personas. High-level scenarios can be identified based on looking further into each phase of the relevant lifecycles. Refer to the User Profiles section for additional information.

II. Lifecycles of Interest

The following diagram depicts the major software lifecycles and the general intersection of these lifecycles. Software built using SDD can exist throughout the depicted lifecycles. In some cases, software remains within a single lifecycle (such as a product built for, and solely used by, a service department). In other cases, software travels through many lifecycles in parallel, such as a product that has previously shipped and is now in active development (for a new version) and sustainment within the originating organization and an ISV.

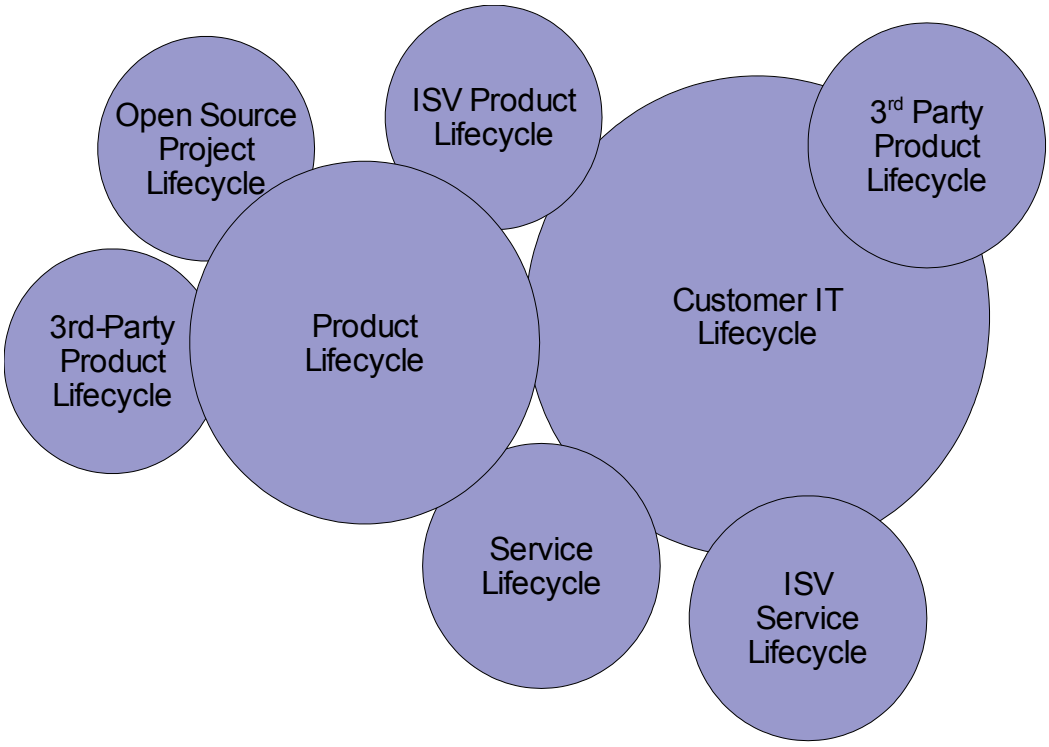


Illustration 1: Software Lifecycles of Interest

Lifecycle	Description
Open Source Project Lifecycle	Open source projects provide some of the componentry found within products. The architecture needs to consider how such projects evolve and how they can impact development and sustainment of products.
Product Lifecycle	Encompasses all internal activities related to development, assembly, test, initial delivery and sustainment of software products. Often under-represented in use cases and scenarios.
ISV Product Lifecycle	The architecture needs to address requirements associated with how ISVs embed and redistribute components.
Third Party Product Lifecycle	In a few cases, a product embeds and preintegrates third party components. In many cases, customers integrate third party products themselves.
Customer IT Lifecycles	The lifecycles pertaining to how IT organizations evaluate, acquire, deploy and maintain software infrastructure and IT development organizations develop, deploy and maintain application services.
Service Lifecycle	Emphasizes corporation's role in supporting and servicing deployments of products through various interface points (service calls, customer visits, IT project)

Lifecycle	Description
ISV Service Lifecycle	Inclusion of this lifecycle helps emphasize the need for ISVs to be able to support and service product deployments that include components from a given corporation.

For purposes of discussion for SDD, the Product Lifecycle and Customer IT Lifecycle are the most interesting in that SDD can meet requirements in these lifecycles, but more importantly many of the requirements in the union of these two lifecycles are also found in other lifecycles. Hence the remainder of this document will focus on the Product lifecycle and Customer IT Lifecycle.

III. Product Lifecycle

The major phases of this lifecycle depict the milestones that software systems go through, from the developer's desktop to the customer's door. It includes:

Product Development – Creation of basic value. What's more important here is that design criteria and design goal decisions made at this stage can have a very good, or very bad, impact on all other phases of the lifecycle.

Product Packaging/ Assembly - This phase is where individual software components are packaged and assembled into aggregate sets of deliverable software. Flaws or ineffeciencies in this phase can quickly be magnified as they impact the rest of the lifecycle.

Test - Testing ensures quality. Difficult testing, lack of coverage, and lack of scalability of testing methods contribute to poor quality.

Deliver - Even after quality components are developed, tested, and packaged, delivery problems can result in large customer dissatisfaction, and contribute to a poor first impression during the later IT planning phases.

IV. Customer IT Lifecycle

The following diagram represents the major phases of the Customer IT Lifecycle. The term "Infrastructure" is inclusive of all components necessary to provide services to business applications. This typically includes middleware such as application or directory services. "Business Application" refers to applications that a customer deploys on top of infrastructure components.

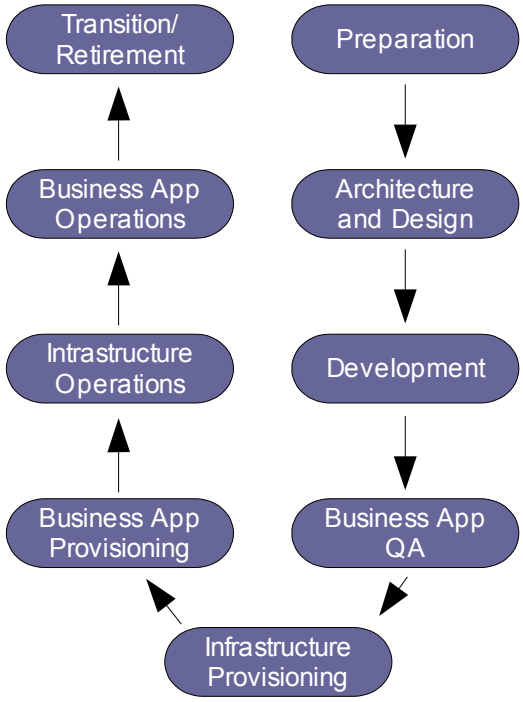


Illustration 2: Customer IT Lifecycle

SDD Use Cases and ultimately requirements stem from various impacts that SDD-flavored technologies can have throughout these phases of the lifecycle. Specific impacted areas in these phases include:

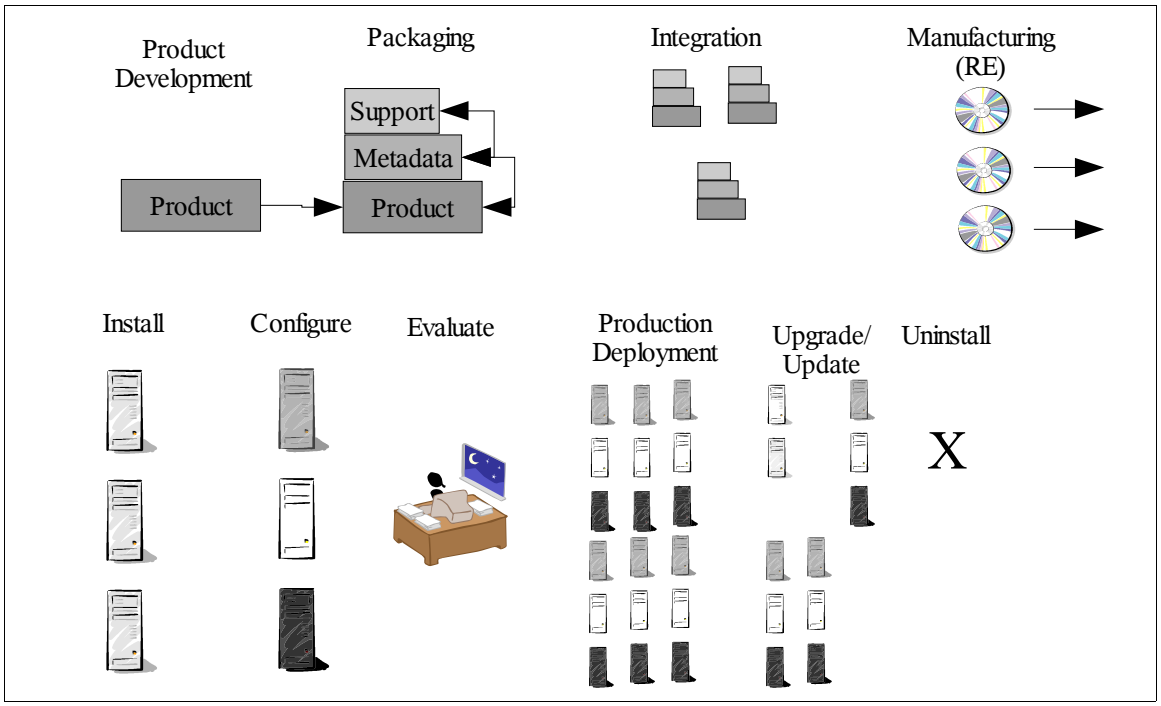
Phase	Areas of Impact
Preparation	• Awareness of Business Requirements • Organization Planning • vendor and Technology Evaluation and Shortlisting • Project Planning and Budgeting
Architecture and Design	• Architecture Design and Review • Prototyping and Proof of Concept • Vendor and Technology Selection
Development	• Business Logic and Process Flows • Integration • Training
Business Application QA	• Provisioning and Operation of QA Infrastructure
Infrastructure Provisioning	• Acquisition and Testing • Creation of 1st Server Image • Provisioning of 2-n Servers
Business Application Provisioning	
Infrastructure Operations	• Monitoring • Defect Reporting and Tracking • Installation of Updates

Phase	Areas of Impact
Business Application Operations	• Business Application Bug Fixes • Release Planning • Release Implementation
Transition and Retirement	

IV. Traditional Software Lifecycle

This method of depicting the software lifecycle is one of many variations on the traditional "build/ship/patch" lifecycle and is a useful way to theme requirements and/or use cases:

An *informal* representation of what most people think the software lifecycle looks like is illustrated below:



All of these phases have attributes which contribute to a successful or unsuccessful product. It is also important to note that the lifecycle starts at the beginning, when the first line of code is written, through the end, when the last piece of executable code is retired. Some high-level features that contribute to the overall experience are outlined below, for the informal lifecycle points illustrated above:

Product Development - Powerful installation and management mechanisms stem from well-defined product content. In other words, the success of the product with respect to its deployment is largely based on the quality and completenessw of the things it delivers and installs.

Packaging – Consistent packaging of content allows consumers to easily know what it is they have on their systems. Packages are the atomic unit of installation to most administrators, and should represent coherent, intuitive subdivisions of the overall product. Also, consistency between platforms eases the learning curve of administrators moving between different platforms.

Integration and RE – The last step before software leaves the "factory" is the bundling of individual components into an end-user-consumable chunk. Given a consistent and complete description of the individual components, integrating them essentially provides the "icing on the cake" for a given product. This is also a key point in the validation of software, as it is the last chance to catch

inconsistencies that may lead to problems further down the line.

Install – Once customers purchase or otherwise acquire software, it is in this phase that the software is actually deployed to hosting environments. This is done through a number of methods today, including manual or automated downloading off of a corporate web presence, purchasing of physical media, or setting up of local cache servers to distribute software internally. However, the goal is always the same, and consistency between the number of avenues of installation can greatly simplify the user experience, and help management of installed software through a similar number of avenues.

Configure - The goal of installation is nearly always to leave installed software on a system or set of systems, ready to be run and already useful. Since running and useful software requires configuration, installers often contain software to decide or specify configuration options and to carry out these configuration steps. Combining the 'install', 'decide' and 'configure' steps results in an installer that proves *monolithic* and inflexible. It is not possible to reuse portions of the installer elsewhere, nor is it possible to perform only the file-system-object portion of the install and leave the configuration to a subsequent step. If, however, these three steps are broken up, many useful operations may be supported, such as factory installs (with later configuration), or serializing and re-using configuration.

Evaluate – In the Evaluation phase, previously installed and configured software is put into operation, in order to evaluate its usefulness. This is typically done for software which is to ultimately be “rolled out” to a number of hosting environments. The software is first executed in a safe, controlled environment, where its behavior can be analyzed and contained. Once the software is evaluated, it is either discarded if it does not meet requirements, or it is prepped for the next phase of deployment (in many cases it goes straight to full-scale deployment).

Production Deployment - Production deployment typically occurs after a product is evaluated in the Eval phase. The deployment phase consists of moving the software out to its production location, initiating the application services, and hooking existing services into the new services. This phase is typically controlled by higher-level provisioning applications. However, in order for these higher-level frameworks to be successful, the lower-level software being deployed must meet certain criteria and expose certain controllable interfaces in order to properly deploy into a given environment.

Upgrade/Update - One of the more difficult and error-prone phases of the lifecycle are software updates. This is because of the literally infinite states that an installation can have before it is upgraded. Any successful upgrade of a product begins at the source of the product and ends with a clear understanding of what happens during an upgrade, what the expected outcome is, and how to determine whether the upgrade was successful or not. Several scenarios within this phase must be represented with supporting use cases and ultimately requirements:

- Backwards compatibility
- Minimized Downtime
- Obsolescence
- Configuration and Data Migration
- Patching/Hotfixes
- New Features

Uninstall - The final phase of the lifecycle is the retiring of the product. This means taking it out of production and removing traces of it from any individual nodes on the network. Uninstall is theoretically defined as the inverse of install. There are many facets of uninstall that must be taken into account, including

- Removal of executable code
- Disposition of configuration and application data
- Dependency resolution