CONTENTS INCLUDE:

Getting Started with Lean Software Development

By Curt Hibbs, Steve Jewett, and Mike Sullivan

ABOUT LEAN SOFTWARE DEVELOPMENT

Lean Software Development is an outgrowth of the larger Lean movement that includes areas such as manufacturing, supply chain management, product development, and back-office operations. Its goal is the same: deliver value to the customer more quickly by eliminating waste and improving quality. Though software development differs from the manufacturing context in which Lean was born, it draws on many of the same principles.

Seven Principles of Lean Software Development

Lean Software Development embodies seven principles, originally described in the book Implementing Lean Software Development: From Concept to Cash[1], by Mary and Tom Poppendieck. Each of these seven principles contributes to the "leaning out" of a software development process.

Eliminate Waste
Waste is anything that does not contribute value to the final product, including inefficient processes, unnecessary documentation, and features that won't be used. Eliminating waste is the guiding principle in Lean Software Development.

Build Quality In
Building quality into a product means preventing defects, rather than using post-implementation integration and testing to detect them after the fact.

Create Knowledge
The knowledge necessary to develop a project, including requirements, architecture, and technologies, is seldom known or understood completely at project startup. Creating knowledge and recording it over the course of the project ensures the final product is in line with customer expectations.

Defer Commitment
Making irreversible decisions at the last reasonable moment allows time for the creation of more knowledge, which results in better decisions. Deferring commitment is positive procrastination.

Deliver Fast
Delivering fast puts the product in front of the customer quickly so they can provide feedback. Fast delivery is accomplished using short iterations, which produce software in small increments by focusing on a limited number of the highest priority requirements.

Respect People
Respecting people means giving the development team's most important resource, its members, freedom to find the best way to accomplish a task, recognizing their efforts, and standing by them when those efforts are unsuccessful.

Optimize the Whole
Optimizing the whole development process generates better results than optimizing local processes in isolation, which is usually done the expense of other local processes.

Lean vs. Agile

Comparing Lean and Agile software development reveals they share many characteristics, including the quick delivery of value to the customer, but they differ in two significant ways: scope and focus. The narrow scope of Agile addresses the development of software and focuses on adaptability to deliver quickly. Lean looks at the bigger picture, the context in which development occurs, and delivers value quickly by focusing on the elimination of waste. As it turns out, they are complementary, and real world processes often draw from both.

GETTING STARTED

Newcomers to Lean Software Development sometimes have trouble implementing a Lean process. The Lean principles don't describe an "out-of-the-box" solution, so one approach is to start with an Agile methodology. However, a number of methodologies exist, and choosing the right one can be difficult.

One Step at a Time

All is not lost. What follows is a set of inter-related practices organized in a step-by-step fashion to allow projects to implement Lean Software Development one step at a time.

Agile Team Management
Made Easier
30,000 Agile Teams
170 Countries
Try It Free
See for yourself why VersionOne is the most utilized agile management tool on the planet
VersionOne.com
VERSIONONE
Simplifying Software Delivery

The following practices can stand-alone, and implementing any of them will have a positive effect on productivity. Lean Software Development relies on prioritization, so the practices are prioritized to generate the highest return on investment. While implementing any one practice will help lean out a process, doing them in order will return the most "bang for the buck."

The list of six practices is preceded by two prerequisites, or "zero practices", every software project should be doing, whether Lean, Agile or something more traditional. If your project doesn't do these things, this is the best place to start.

ZERO PRACTICES

Source Code Management and Scripted Builds are prerequisites for other practices outlined here. They are referred to as zero practices because they need to be in place before taking the first step toward Lean Software Development.

Source Code Management

Source code management (SCM) is a shared repository for all artifacts needed to build the project from scratch, including source code, build scripts, and tests. SCM maintains the latest source code so developers and build systems have up-to-date code.

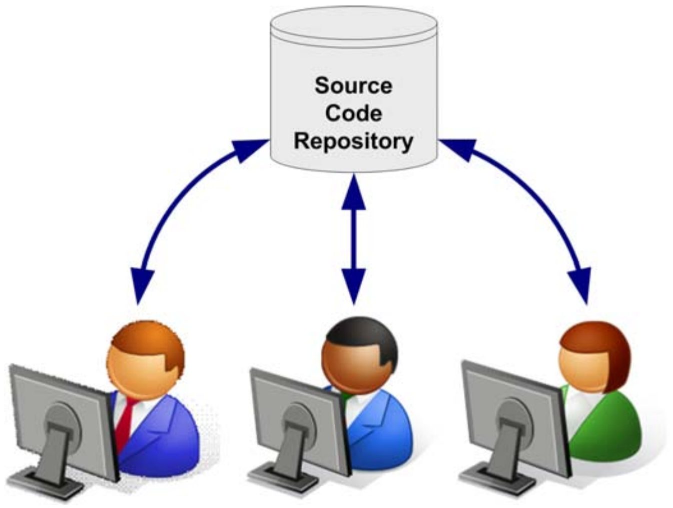

Figure 1: Centralized Repository

Source code management is the first practice described because it is the foundation for a practical development environment, and it should be implemented before going any further.

- Select an appropriate SCM system. Subversion is a popular open source option. Git is a newer distributed SCM system useful for large projects and distributed teams.
- Put everything needed to build the product from scratch into the SCM system so critical knowledge isn't held only by specific individuals.

Scripted Builds

Scripted builds automate a build process by executing a set of commands (a script) that creates the final product from the source code stored in SCM. Scripts may be simple command files, make files, or complex builds within a tool such as Maven or Ant.

Scripted builds eliminate the potential errors of manual builds by executing the same way each time. They complete the basic development cycle of making changes, updating the SCM repository, and rebuilding to verify there are no errors. Select an appropriate build tool for your project. Integrated development environments like Visual Studio or Eclipse have build managers or integrate with 3rd party build managers. Create a script that builds the product from scratch, starting with source code from SCM.

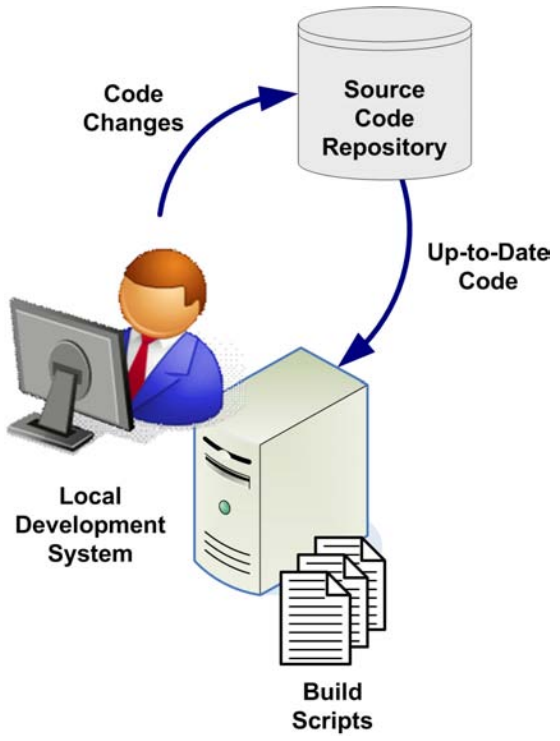

Figure 2: Zero Practices

Lean Principles

- **Create Knowledge:** SCM consolidates project knowledge in a single place.
- **Eliminate Waste:** Manual work is eliminated by automating builds.
- **Build Quality In:** Automating builds eliminates a source of errors.

DAILY STANDUP

Daily standup meetings allow each team member to provide status and point out problems or issues. The meetings are short and not intended to resolve problems, rather they serve to make all team members aware of the state of the development effort.

Borrowing from the Scrum methodology, standups are conducted by having each member of the team answer three questions:

What did I do yesterday?

What will I do today?

What problems do I have?

Effective daily standups result from adhering to several simple rules:

- Require all team members to attend. Anyone who cannot attend must submit their status via a proxy (another team member, email, etc.).
- Keep the meeting short, typically less than 15 minutes. Time-boxing the meeting keeps the focus on the three questions.
- Hold the meeting in the same place at the same time, everytime.
- Avoid long discussions. Issues needing further discussion are addressed outside the meeting so only the required team members are impacted.

Hot Tip The Japanese word *tsune* roughly translated means "daily habits." It refers to things such as taking a shower that are so ingrained into a daily routine that skipping them leaves one feeling that something is missing[2]. Make the daily standup part of the team's *tsune* so that a day without a standup feels incomplete.

Lean Principles

- **Respect People:** Standups foster a team-oriented attitude; team members know what other members are doing and can get or give help as needed to move the project forward.
- **Create Knowledge:** Sharing information regularly creates group knowledge from individual knowledge.

AUTOMATED TESTING

Automated testing is the execution of tests using a single command. A test framework injects pre-defined inputs, validates outputs against expected results, and reports the pass/fail status of the tests without the intervention of a human tester. Automated testing ensures tests are run the same way every time and are not subject to the errors and variations introduced by testers.

While automated testing can be applied to all types of testing from unit and integration tests to user acceptance and performance/load tests, unit and integration testing is the best place to start.

- Identify an appropriate test framework for the language in use. JUnit (for Java) and NUnit (for Microsoft .NET languages) are common frameworks.
- Require all new code modules to have a unit test suite before being included in the build.
- Retrofit unit test suites to existing legacy code only when the code is modified (writing unit tests for code which is already written and functional usually is not cost effective).
- Develop integration tests by combining code modules and testing the modules together.
- Use stubs and mock objects to stand in for code which has not yet been developed.

Hot Tip Developing automated tests alongside production code may be a paradigm shift for many developers. One way to help developers adjust is to define testing standards calling out both what to test and how to do it. Adherence to the standards will create a culture where automated tests are the norm and will pay off in higher quality software.

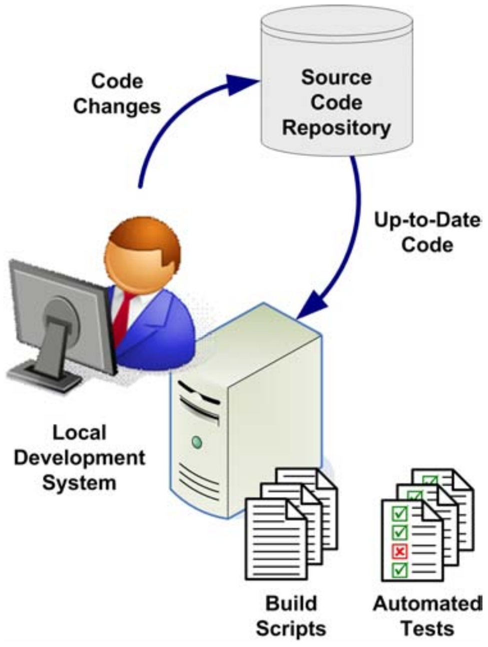

Figure 3: Automated Testing

Test Execution

Each developer runs unit tests on individual code modules prior to adding them to the source code repository, ensuring all code within the repository is functional. An automated build runs both unit and integration tests to ensure changes do not introduce errors. The next practice, continuous integration, will make use of the build scripts and test suites to test the entire system automatically each time changes are checked into the repository.

Lean Principles

- **Build Quality In:** Automated tests executed regularly and in a consistent manner prevent defects.
- **Eliminate Waste:** Defects detected early are easier to correct and don't propagate.
- **Create Knowledge:** Tests are an effective way to document how the code functions.

CONTINUOUS INTEGRATION

Continuous integration (CI) is the frequent integration of small changes during implementation. It seeks to reduce, or even eliminate, the long, drawn-out integration phase traditionally following implementation. Integrating small changes doesn't just spread the effort out over the whole cycle, it reduces the amount of integration time because small changes are easier to integrate and aid debugging by isolating defects to small areas of code.

CI systems use a source code repository, scripted builds, and automated tests to retrieve source code, build software, execute tests, and report results each time a change is made.

- Use a dedicated build machine to host the CI system. Refer to the Continuous Integration: Servers and Tools Refcard (#87) for details on setting up a CI system.
- Check code changes into the repository a minimum of once a day (per developer); once an hour or more is even better.
- Immediately address any failures in the build. Fixing the build takes precedence over further implementation.

Hot Tip

While the use of a dedicated computer, or build machine, to host the CI system may seem obvious for a large project, it provides advantages on small projects as well:

- Dedicated machines don't compete for resources, so builds are quicker and the results get back to the developers sooner.
- Dedicated machines have a stable, well-known configuration. Builds don't fail because a new version of a library was loaded or the runtime environment was changed.

A CI system can also check coding standards, analyze code coverage, create documentation, create deployment packages, and deploy the packages. Anything that can be automated can be included in a CI system.

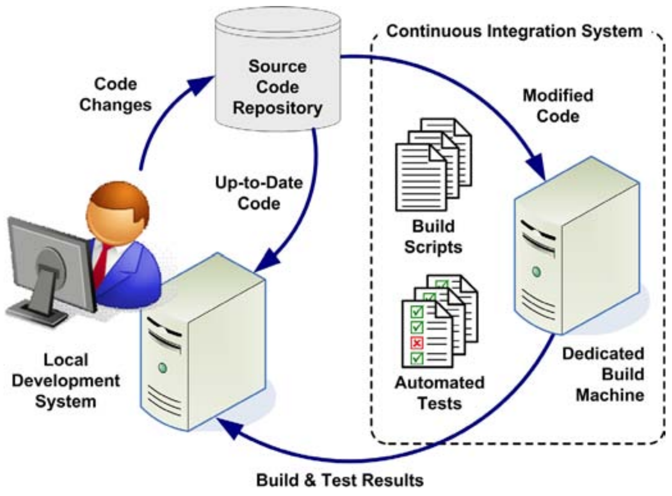

Figure 4: Continuous Integration

Lean Principles

- **Build Quality In:** Continuous build and test ensures code is always functional.
- **Eliminate Waste:** Frequent, small integrations are more efficient than an extended integration phase.

LESS CODE

Less code is not about writing less software, it's about implementing required functionality with a minimum amount of code. Large code bases mean more implementation, integration, and debugging time, as well as higher long term maintenance costs. All of these are non-value added work (i.e., waste) when the code base contains unneeded or inefficient code.

All aspects of software development can affect the code base size. Requirements analysis resulting in features with little likelihood of use and overly generic, all-encompassing designs generate extra code. Scope creep and unnecessary features increase the amount of code. Even testing can generate unnecessary code if the code under test is itself unnecessary.

Minimizing code base size requires two actions: identify and eliminate unnecessary code, and write efficient code. Minimizing code base size is not unlike a fitness program: diet to eliminate the excess, and exercise to shape up what's left.

Eliminate Unnecessary Code

Eliminating unnecessary code means identifying the code, or the forces that create it, and removing it.

- Adopt a fierce, minimalist approach. Every bit of code added to the code base must be justifiable. Remove excessive requirements, simplify designs, and eliminate scope creep.
- Reuse code and employ libraries to reduce the amount of new code that must be written.
- Prioritize requirements so developers implement important features first. As customers adjust the priorities over the course of development, they drive development of only useful features; unused features never get implemented.
- Develop only for the current iteration. Working too far ahead risks doing work that will be thrown away as requirements and design change over time.

Improve Code Efficiency

Code efficiency doesn't refer to creating small, compact code by using arcane tricks and shortcuts. In fact, the opposite is true; efficient code uses coding standards and best practices.

- Use coding standards to write readable and understandable code. Use best practices and proven techniques that are well understood by other developers.
- Develop flexible, extensible code. Design and implement code with design patterns, refactoring, and emergent design.

Hot Tip

The "big design up front", or BDUF, approach to design can lead to overdesign and unused code. The opposite approach, sometimes referred to as "you ain't gonna need it" or YAGNI, creates only what is needed at the moment, but it can lead to brittle designs and inefficient code. A compromise that creates only what currently is necessary, but tempers that with some thought for the future, is a better approach. Scott Bain's book *Emergent Design*[2] describes such an approach.

Lean Principles

- **Eliminate Waste:** Frequent, small integrations are more efficient than an extended integration phase.

- **Build Quality In:** Automated tests executed regularly and in a consistent manner prevent defects.

SHORT ITERATIONS

Iterations are complete development cycles resulting in the release of functional software. Traditional development methodologies often have iterations of six months to a year, but Lean Software Development uses much shorter iterations, typically 2 to 4 weeks. Short iterations generate customer feedback, and more feedback means more chances to adjust the course of development.

Feedback and Course Corrections

Feedback from the customer is the best way to discover what's valuable to them. Each delivery of new functionality creates a new opportunity for feedback, which in turn drives course corrections due to clarification of the customer's intent or actual changes to the requirements. Short iterations produce more feedback opportunities and allow more course corrections, so developers can hone in on what the customer wants.

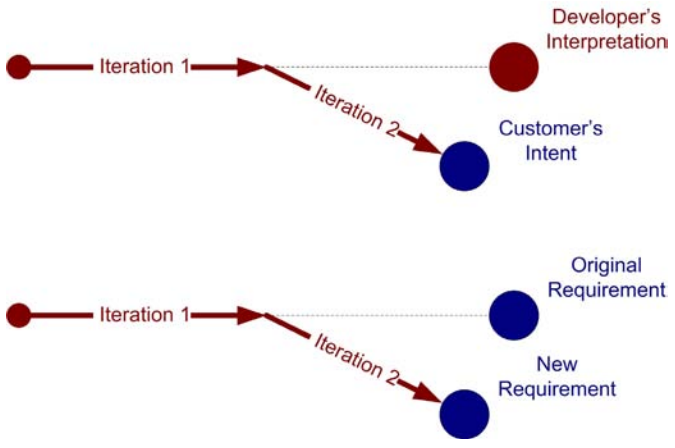

Figure 5: Course Corrections

Using Short Iterations

Several techniques aid in the implementation of a process using short iterations:

- Work requirements in priority order. High priority requirements typically are well-defined, easiest to implement, and provide the most functionality in a short period of time.

- Define a non-negotiable end date for the iteration; sacrifice functionality to keep the team on schedule. End dates focus the team on delivering the required functionality. Even if some features are not completed, delivering those that are ensures customers get new functionality on a regular basis.

- Mark the end of the iteration with a demo and an official handoff to the customer. Demos foster pride in the product by allowing the team to show off its work.

- Deliver the product to the customer, whether it's a ready-for-deployment application or an interim release. Getting the product in the customer's hands for in-depth evaluation is the best way to generate feedback.

Hot Tip Teams struggling to complete iterations successfully are often tempted to lengthen their iterations; however, longer iterations tend to hide problems. Instead, struggling teams should reduce the iteration length, which reduces the scope, focuses the team on a smaller goal, and brings impediments to the surface more quickly so they can be resolved.

Lean Principles

- **Eliminate Waste:** Frequent, small integrations are more efficient than an extended integration phase.

- **Deliver Fast:** New, functional software is delivered to the customer in closely-spaced intervals.

CUSTOMER PARTICIPATION

Customer participation in traditional projects typically is limited to requirements specification at the beginning of the project and acceptance testing at the end. Collaboration between customers and developers in the intervening time is limited, typically consisting of status reports and the occasional design review.

Lean Software Development approaches customer participation as an on-going activity spanning the entire development effort. Customers write requirements and developers produce functional software from those requirements. Customers provide feedback on the software and developers act on that feedback, ensuring developers are producing what the customer really wants.

Involve the Customer

Key to establishing effective customer collaboration is involving the customer in the entire development process, not just at the beginning and end. Engaging the customer, reporting status, and providing a feedback path all help keep the customer involved.

- Engage the customer by having them write and prioritize the requirements. Customers get a sense of ownership, and they can direct the course of development.

- Have the customers write the acceptance tests (or at least specify their content), and, if possible, run the tests as well. Involvement in testing the product allows customers to specify exactly what it means to satisfy a requirement.

- Provide useful, easily-accessible status. For example, a list of the requirements in work and the status of each. Include status on problems affecting development to avoid surprises.

- Provide access to the product so the customer can see for themselves how it works, and provide a simple, direct feedback path so customers can input feedback easily.

Collaborate

Collaborating directly with the customer is necessary for developers to refine the requirements and understand exactly what the customer wants.

- Designate a customer representative. The representative writes and/or collects requirements and prioritizes them. The representative clarifies requirements for developers.
- Schedule face-to-face time with the customer. At the very least, include a demo at the end of each iteration.

Hot Tip Actual customers make the best customer representatives, but when customer representatives are not available a **customer proxy** can fill the role. A customer proxy should be from the development team's organization and must have a good understanding of the customer's needs and business environment.

Lean Principles

- **Create Knowledge:** Through collaboration, requirements are discovered and refined over time.
- **Defer Commitment:** Involving customers throughout the process eliminates the need to make decisions up front.

SUMMARY

Most discussions of Lean Software Development don't define specific practices for implementing the process, and the large number of Agile methodologies to choose from can leave newcomers confused and uncertain where to start. The specific practices outlined here provide a step-by-step approach to implementing a Lean Software Development process. Adopt one, several, or all the practices and take your first step into the world of Lean Software Development.

References

[1]Implementing Lean Software Development: From Concept to Cash, Poppendieck/Poppendieck, Addison-Wesley Professional, 2006

[2]Moving Toward Stillness, Lowry, Tuttle Publishing, 2000.

[3]Emergent Design: The Evolutionary Nature of Professional Software Development, Bain, Addison-Wesley Professional, 2008

Some of the concepts and material in this Refcard were adapted from The Art of Lean Software Development, Hibbs/Jewett/Sullivan, O'Reilly Media, 2009.

ABOUT THE AUTHORS

Curt Hibbs co-leads the Boeing team responsible for the adoption of Lean and Agile software engineering practices across Boeing's Defense, Space & Security business unit. He has been a software engineer for 30+ years, and during that time he has done just about everything related to developing software products, from working for WordStar, Hewlett Packard, the C.I.A, and more, to being the CTO of several startups. He has worked for Boeing since 2003.

Steve Jewett is a software developer with the Boeing Company, where he is involved in the development of cognitive decision support systems. Over a 25 year career he has developed software for automated test equipment, weapon/aircraft integration, embedded systems and desktop and web applications. He currently leads an agile software development team and works to promote Lean-Agile software development at Boeing.

Mike Sullivan has over 6 years of experience teaching at the university level, and has spent the last 5+ years working with software teams in small companies and large corporations to drive valuable solutions and improve team dynamics. He is currently working in a small research team within a large corporation, implementing Lean techniques to improve the software his team delivers.

ISBN-13: 978-1-934238-74-5
ISBN-10: 1-934238-74-0
50795
9 781934 238745
$7.95

Version 1.0