ISSN No. 2395-0900
IJESTA International Journal of Engineering Studies and Technical Approach

Identifying Wastes in Software

Mr. Piyush Kumar Pareek* & Dr. A.N.Nandakumar **

****Assistant Professor, Department of CSE, Kammavari Institute of Technology, Bengaluru.***

*****Principal , R.L. Jalappa institute of Technology* , Doddaballapur, Visvesvaraya Technological University, *Belgaum***

ABSTRACT:

The concept of "waste" in business was coined in the manufacturing industry during late 1940s. It was used by Toyota. In those days, automobiles involved a huge amount of manufacturing, and companies had to charge their customers a high price. The only option for Toyota to reduce car prices was to find ways to reduce the manufacturing costs. As part of this exercise, they started identifying "waste," which meant the feature (or process step) that did not add value for the customer. Once they identified the waste, they created ways to eliminate that waste from the system. In this Paper we discuss various wastes in software through Literature survey.

Keywords: Waste, system & Toyota.

INTRODUCTION:

Lean development is a culture; waste reduction is one of the results. Removing waste can improve operational efficiency, but more importantly, it can reduce the development cycle and increase customer value. Shorter cycles can improve innovation, competitiveness, and responsiveness in the marketplace. They can also provide a valuable opportunity for learning and continuous improvement for the development teams.

Waste is, in fact, the opposite of value (a capability delivered to the customer through which the customer attains a tangible or intangible benefit). So whatever feature or functionality or process step that neither adds value nor is used will be considered waste and should be eliminated from the system/product/process.

LITERATURE SURVEY:

ISSN No. 2395-0900
International Journal of
IJESTA Engineering Studies and Technical Approach

7 Wastes of Software Development

1. Partially Done Work :- Partially done work often becomes obsolete long before it's ever finished. Until software code is released into production, you have no clue whether it solves the business problem or not.
2. Extra Features:- Occasionally a product stakeholder will have a "pet feature" that s/he insists will be included in the product.
3. Relearning:- Poorly written or undocumented code leads to untold amounts of relearning.
4. Handoffs:- Simply try to reduce the number of handoffs in SDLC and Find ways to integrate teams need to work together.
5. Delays: - Waiting for appropriate resources to become available to start working on a project
6. Task Switching: - Eliminate unimportant work and interruptions! If it isn't delivering value, stop doing it
7. Defects: - That defects are a source of software development waste should be self-evident, but exactly how do we measure the amount of waste created by a single defect?

Partially done work is probably the biggest killer of all the wastes. Partially done work is essentially work-in-progress. Until this work is done, you don't know that there are quality issues i.e. you don't know that the customer will be happy. You don't know if there are going to be problems one you deploy your software onto your production systems. So the idea should be to complete work-in-progress as soon as possible i.e. minimize work-in-progress as much as possible. Examples of partially done work are:

Code that is completed but not checked in to your version control system - if it's not checked in, you don't know if your code changes are going to break the build

- ***undocumented code*** - If your code is undocumented, if the developer leaves and someone has to take over, there's going to take longer for the developer to get up to speed. Additionally, if bugs are found, it will be harder for the original developer to figure out what he has done.

- ***untested code (both unit tests and functional tests)*** - if your code is untested, you won't know till the code is in your customers hands that there is a bug. The further downstream you are in the process the more costly it's going to be to fix the bugs. So if you build quality in from the start (like writing unit tests) you'll find out the moment you execute the tests

- ***code that exists on your staging environment and not your production environment*** - I hear this all the time - "works on my machine" - enough said. Only once you're on production can you be sure the software is 100%. Production always surfaces issues, so the sooner you get it on production servers, the better.

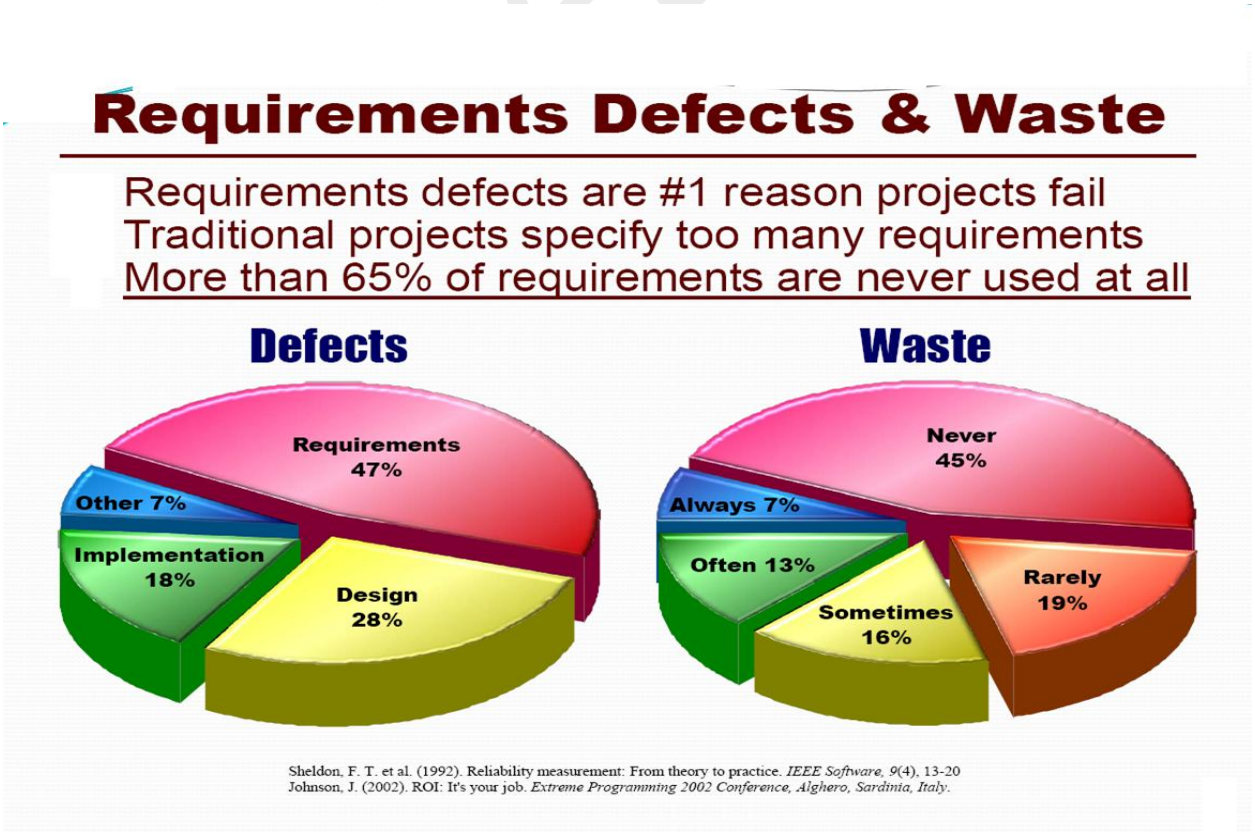
- ***code that is commented out*** - makes the software less readable and maintainable [1]

ACCORDING TO ADAGE TECHNOLOGIES WASTES IN SOFTWARE ARE AS FOLLOWS:

Waste #1 - Unnecessary Features

In a traditional waterfall development approach, the requirements team is responsible for identifying all of the required features of the application. Since the requirements team will only have this one chance to identify all of the necessary features, the requirements team will spend too much time trying to solve all of the problems. This approach has lead to not only inefficiencies in utilization of resources, but also to adding features that no one really needs or wants.

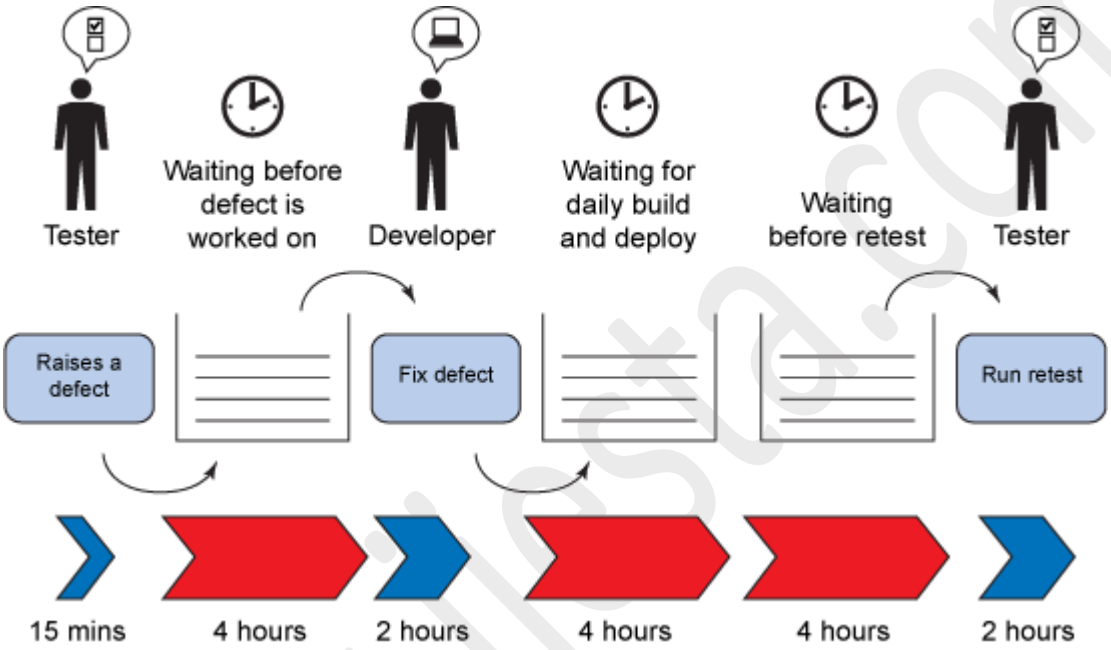
The Standish Group has some very widely accepted reports that 45% of implemented features are "never used" and that another "15%" are "rarely used". Adage believes this occurs because in a waterfall methodology there is too much speculation as to what the client will want. Adage's approach is more of an "on demand" approach. By utilizing an Agile approach, we can address the most needed features first. In Lean terms, Adage's approach is a pull structure as opposed to waterfall which is more push.



ISSN No. 2395-0900
International Journal of
Engineering Studies and Technical Approach
IJESTA

Waste #2 - Waiting (for requirements, testing, etc)

The goal for Lean and Agile is to get a working product to customer as efficiently as possible. The more time waiting for requirements or approval means wasted utilization for members of the development team. So to remove the cost of waiting, Adage espouses smaller development cycles. It allows the development team to perform overlapping efforts. The business owners can identify the next set of features while the development and QA teams implement the last requirements. Breaking big projects into smaller cycles (Sprints) is a major tenet of Agile development.



Waste #3 - Decentralized Development Team

By centralizing our development team to one location, Adage removes the waste of either having to make phone calls, send emails, IM, conference calls to work through a problem. While usefull, email and IM is not always the most efficient when trying to work through a problem. Being able to turn around to the person behind and get a question answered saves minutes if not hours of productivity.

Waste #4 - Gold Plating

In software development, there is not good enough, good enough, and too good. The focus is to build as high a quality of a product as is cost effective. Adage focuses on making sure that the quality of the application that we develop is within the range of good enough. This may seem like a bad thing, but you must consider that most software efforts are not really ever done. So, whatever you create today may be discarded tomorrow. So if you overspend your time making a feature TOO perfect (aka Gold Plating), you might be just wasting more time making some perfect that will not survive.

ISSN No. 2395-0900
International Journal of Engineering Studies and Technical Approach
IJESTA

Waste #5 - Scope Too Big for Resources/Sprint

Too often people try to identify all of the features an application should have at the outset of the development process. The goal should be to define a roadmap, which is a very loosely identified set of requirements. From this roadmap, the requirements should be focused down to a list of requirements that the development team can accomplish within a few cycles. If the requirements team spends too much time working on the entire set of requirements, then the development team sits in a wait state. Additionally, when the scope is too big for the development team, then the scope is too big for the requirements team. By reducing the focus to a small subset of features, the requirements team will usually provide better requirements.

Waste #6 - Underutilizing Available Toolsets or Patterns

In recent years, patterns have developed to identify common problems. These patterns are incredibly useful in not trying to recreate the wheel. Additional toolsets that can be utilized to remove "unnecessary movement" are control libraries and code generation tools. Adage takes advantage of both. We use Telerik and Infragistic control libraries as well as a code generation tool that assists in manufacturing our data access layer and business objects in the form of CSLA.Net.

Waste #7 - Defects

How to reduce defects? It's the million dollar question in software development. Testing is usually the answer. But the Lean philosophy teaches us that quality is the goal for defect reduction. And therefore since quality needs to be built in, that quality cannot be inspected in, we come to the conclusion that quality of code must be built in. Adage addresses this through two means: code reviews and unit testing. This starts with the developer owning responsibility and not relying on a QA person to catch the problem. QA starts in development, not after the developer thinks it is done and pushes it to QA.

Waste #8 - Unused Employee Creativity

This form of waste is the most prevalent in most waterfall methodologies. It is the identification of this waste that truly validates Agile as a complete solution. In waterfall methodologies, programmers are treated like a unit of work, a cog. Programmers are given a specification and are expected to code to that specification. But they are generally not given the context of the specification or how it relates to other features that are being asked of them. It should be expected that at any time a programmer should be able "stop the line" and ask for clarification if the requirement is incongruent with others or if there might be a more efficient solution. In waterfall, this is an expensive activity, because of the overload of project management. In Agile this is anticipated, expected and encouraged [2]

Software maintenance is a key activity in software development requiring considerable effort and time. Hence, it is important to increase the efficiency and effectiveness of the maintenance process. The objective of this article is to introduce a palette of indicators to assess the maintenance process based on indicators lean indicators. Four indicators aiming at detecting waste have been proposed, namely the inflow of maintenance requests, the flow of

maintenance requests through the maintenance process with regard to continuous value creation and high throughput, the analysis of lead-times, and the analysis of workload. [3]

Software is a pro duct that is purchased either as a standalone product like a game or a computer program, or as a part of another pro duct embedded in hardware. Most useful software is embedded in something larger than its code base. The software development is in other words a sub process of the product development in which the software is embedded. Software changes continually and modifying production software tend to add complexity and increase expenses.

Lean thinking enables companies to define value, map value-creating steps and perform these more effectively [5]

CONCLUSION

Value chains are a technique to help teams and organizations develop an understanding of how long tasks take and how much waste is present. Value chains can help a team understand when the work product is not being actively worked on, and by removing the idle times, teams can achieve dramatic reductions in cycle time. For most organizations, the key value chain is how long it takes to get an idea into production. Much waste can be attributed to waiting, in particular, these common forms:

- Waiting for infrastructure
- Waiting for applications to be deployed
- Waiting for other teams
- Waiting for reviews to complete

Using a cloud for development and test makes it possible for teams to rapidly provision infrastructure, on demand, to test their applications. This approach can reduce waiting from days and weeks to minutes. Teams can have access to production-like environments much earlier in the development and testing phases [4] .

REFERENCES:

i. ETechnology Management , The 7 Software Development Wastes by COO and Scrum Master, Jack Milunsky
ii. http://www.adagetechnologies.com/blog/patrick-emmons/get-lean-avoid-8-wastes-in-development/
iii. Kai Peterson , “ Palette of Lean Indicators to Detect Waste in Software Maintenance: A Case Study “ , Agile Processes in Software Engineering and Extreme Programming
iv. Lecture Notes in Business Information Processing Volume 111, 2012, pp 108-122 .
v. http://www.ibm.com/developerworks/rational/library/leaner-software-development-with-the-aid-of-collaborative-lifecycle-management/leaner-software-development-with-the-aid-of-collaborative-lifecycle-management-pdf.pdf
vi. Wang et all, "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development ,Free University of Bozen/Bolzano, Italy