

Lean Software Development

Sushant Kumar

07030244024

SDM 07-09

Symbiosis Center For Information Technology

History of Lean Thinking and Lean Software Development

- Toyota has started in the 1980s to revolutionize the automobile industry with their approach of "Lean Manufacturing"
 - ❖ to eliminate waste
 - ❖ to streamline the value chain (even across enterprises)
 - ❖ to produce on request (just in time), and
 - ❖ to focus on the people who add value.
- Lean Thinking capitalizes on the intelligence of frontline workers, believing that they are the ones who should determine and continually improve the way they do their jobs.
- Mary and Tom Poppendieck have transferred principles and practices from the manufacturing environment to the software development environment.

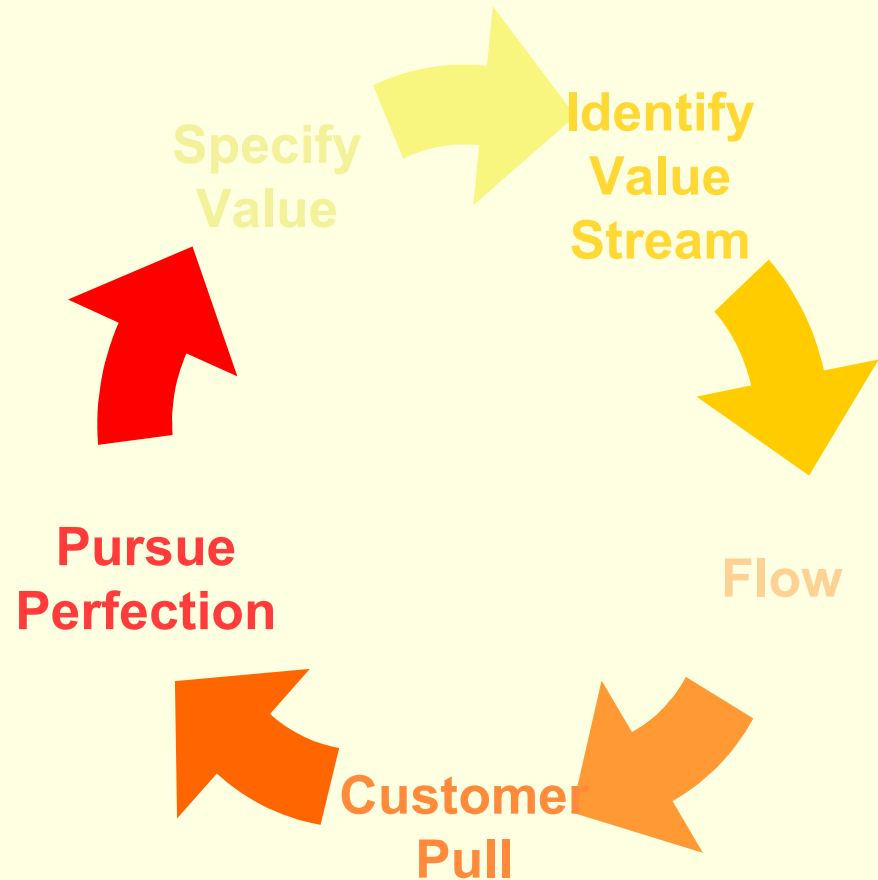
Manifesto for Agile Software Development

- **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.

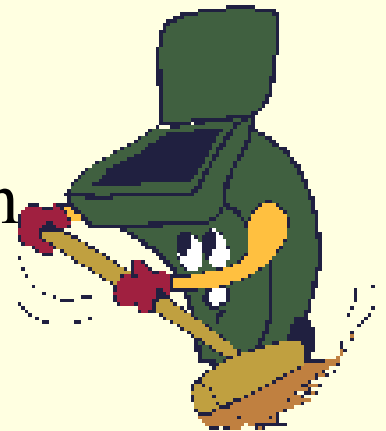
Principles of Lean Thinking

1. Eliminate Waste
2. Increase Feedback
3. Delay Commitment
4. Deliver Fast
5. Build Integrity In
6. Empower the Team
7. See the Whole



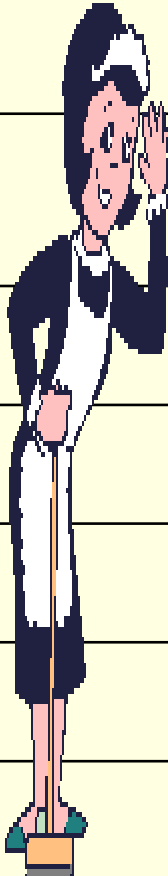
Principle #1: Eliminate Waste

- Does not mean to throw away all documentation, but to spend time only on what adds real customer value.
- Thus, the first step to implementing lean development is learning to see waste.
- The second step is to uncover the biggest sources of waste and eliminate them.

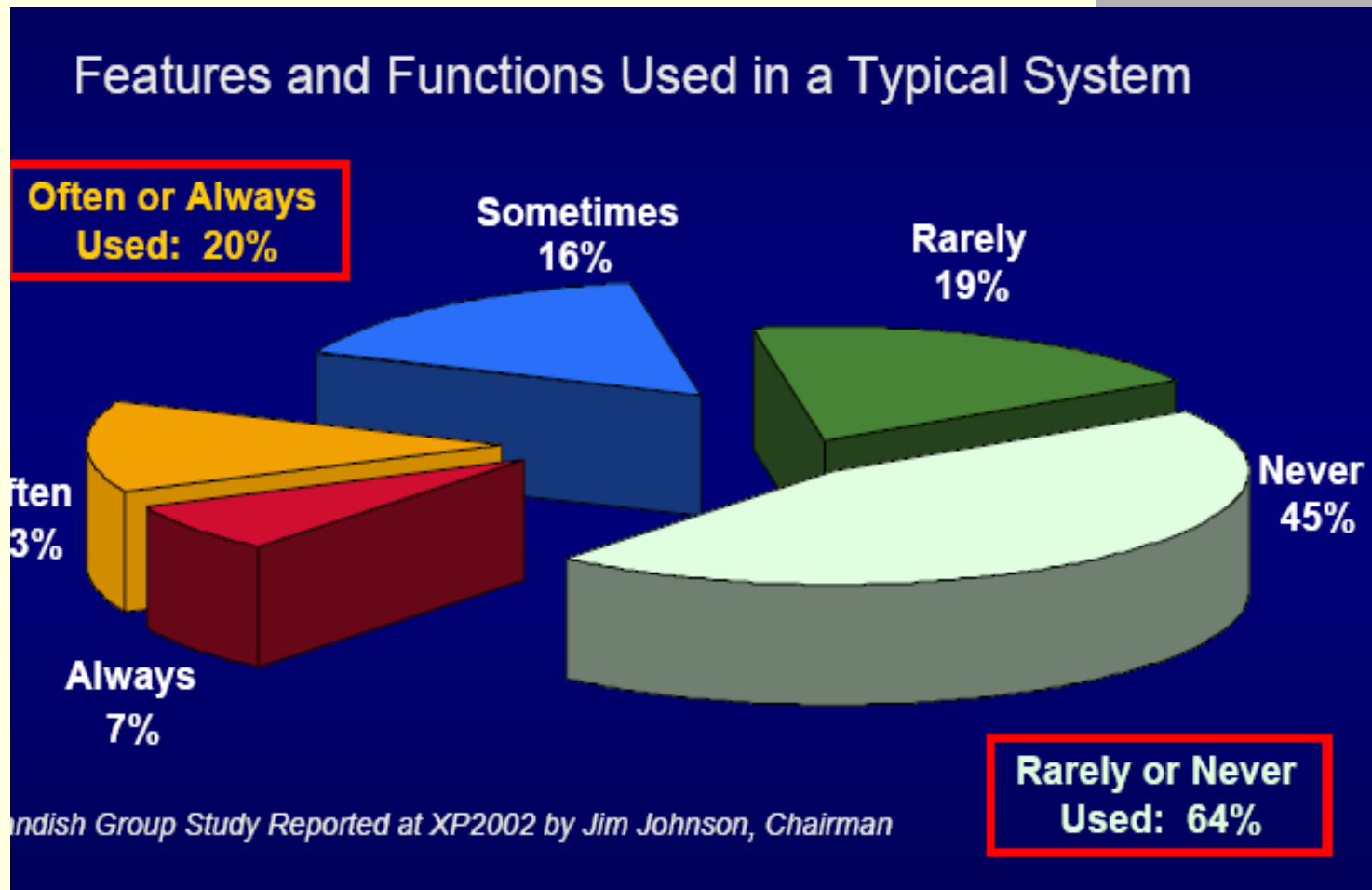


Seeing Waste

Wastes of Manufacturing	Wastes of Software Development
Inventory	Partially work done
Extra processing	Paperwork or excess documentation
Overproduction	Extra features
Transportation	Building the wrong thing
Waiting	Waiting for the information
Motion	Task switching
Defects	Defects



The biggest source of waste



Eliminate waste By using Traditional Value Stream

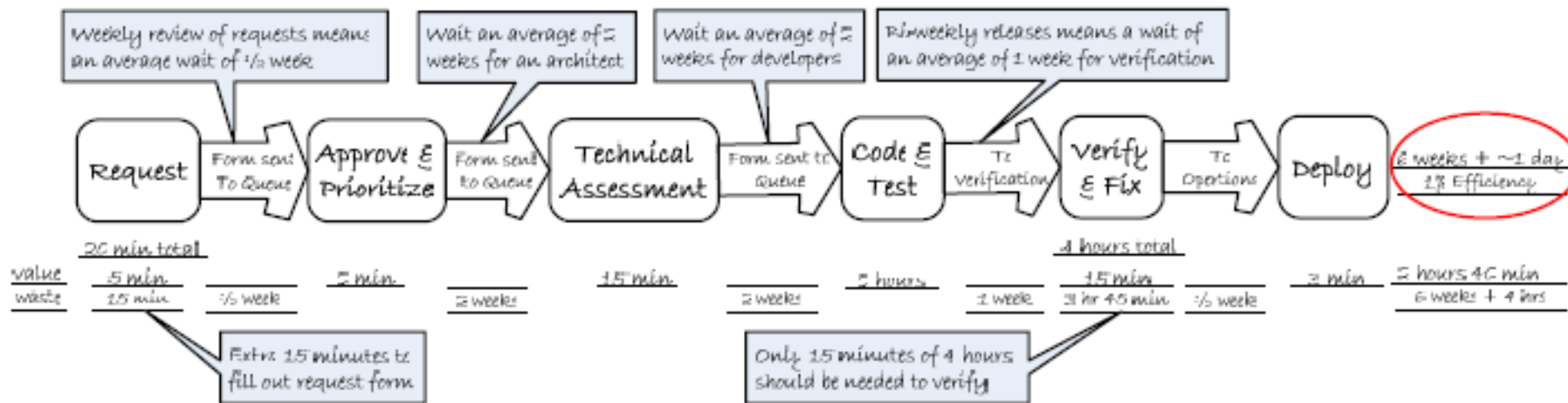


Example 1

Of course there is a developer available

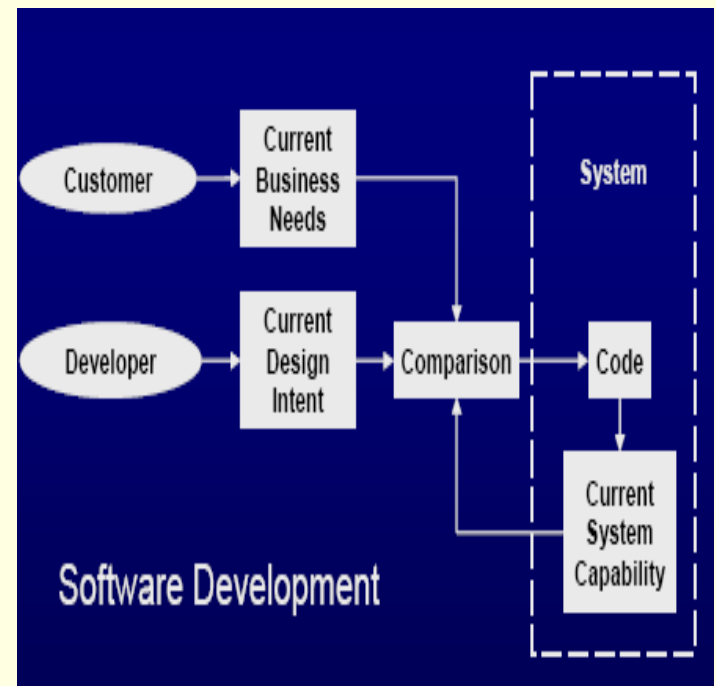


Example 2



Principle #2: Amplify Learning

- Does not mean to keep on changing your mind, but to increase feedback, when you have tough problems.
- When a problem develops...
 - ❖ the first thing to do is to make sure the feedback loops are all in place,
 - ❖ the next thing to do is to increase the frequency of the feedback loops in the problem areas.



Amplify Learning with Synchronization

- Whenever several individuals are working on the same thing, a need for synchronization occurs. The need for synchronization is fundamental to any complex development process.
- Synchronize / integrate technically:
 - ❖ Integrate daily within a team (i.e. check-in at least daily into local repository)
 - ❖ Integrate weekly across multiple teams (i.e. check-in at least weekly into the central repository)

Principle #3: Decide as Late as Possible



- Does not mean to procrastinate, but to keep your options open as long as practical, but no longer.
- We usually don't give our customers the option to change their minds. And yet, almost everyone resists making irrevocable decisions in the face of uncertainty. Options allow fact-based decisions based on learning rather than speculation.
- Premature design commitment restricts learning, exacerbates the impact of defects, limits the usefulness of the product, and increases the cost of change.
- But, options are not free and it takes expertise to know which options to keep open.

Principle #4: Deliver as Fast as Possible



- Does not mean to rush and do sloppy work, but to deliver value to customers as soon as they ask for it.
- Customers like rapid delivery, which often translates to increased business flexibility. Companies can deliver faster than customers can change their minds. Companies have fewer resources tied up in work-in progress.
- The principle *Deliver as Fast as Possible* complements *Decide as Late as Possible*: The faster you can deliver, the longer you can delay decisions.

Real world examples

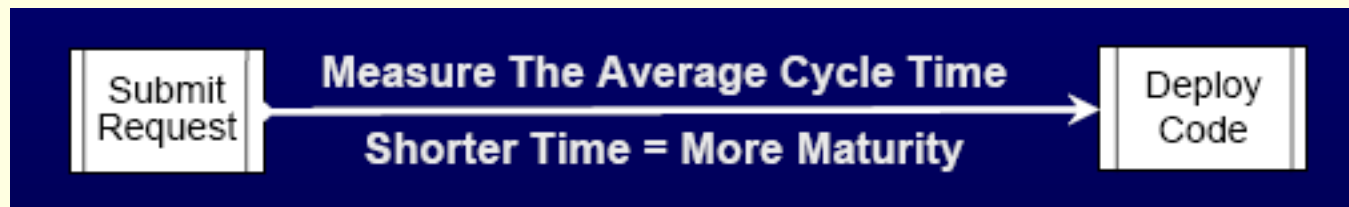
- The most disciplined organizations are those that respond to customer requests

- ❖ Rapidly
- ❖ Reliably
- ❖ Repeatedly



- Software Development Maturity

- ❖ The speed at which you reliably and repeatedly convert customer requests to deployed software



Principles of Speed

- Pull from customer demand
 - ❖ Pull with an order
 - ❖ Don't push with a schedule
- Make work self-directing
 - ❖ Visual Workplace
- Rely on local signaling and commitment
 - ❖ Kanban
 - ❖ Scrum Meetings
- Use Small Batches
 - ❖ Limit the amount of work in the pipeline

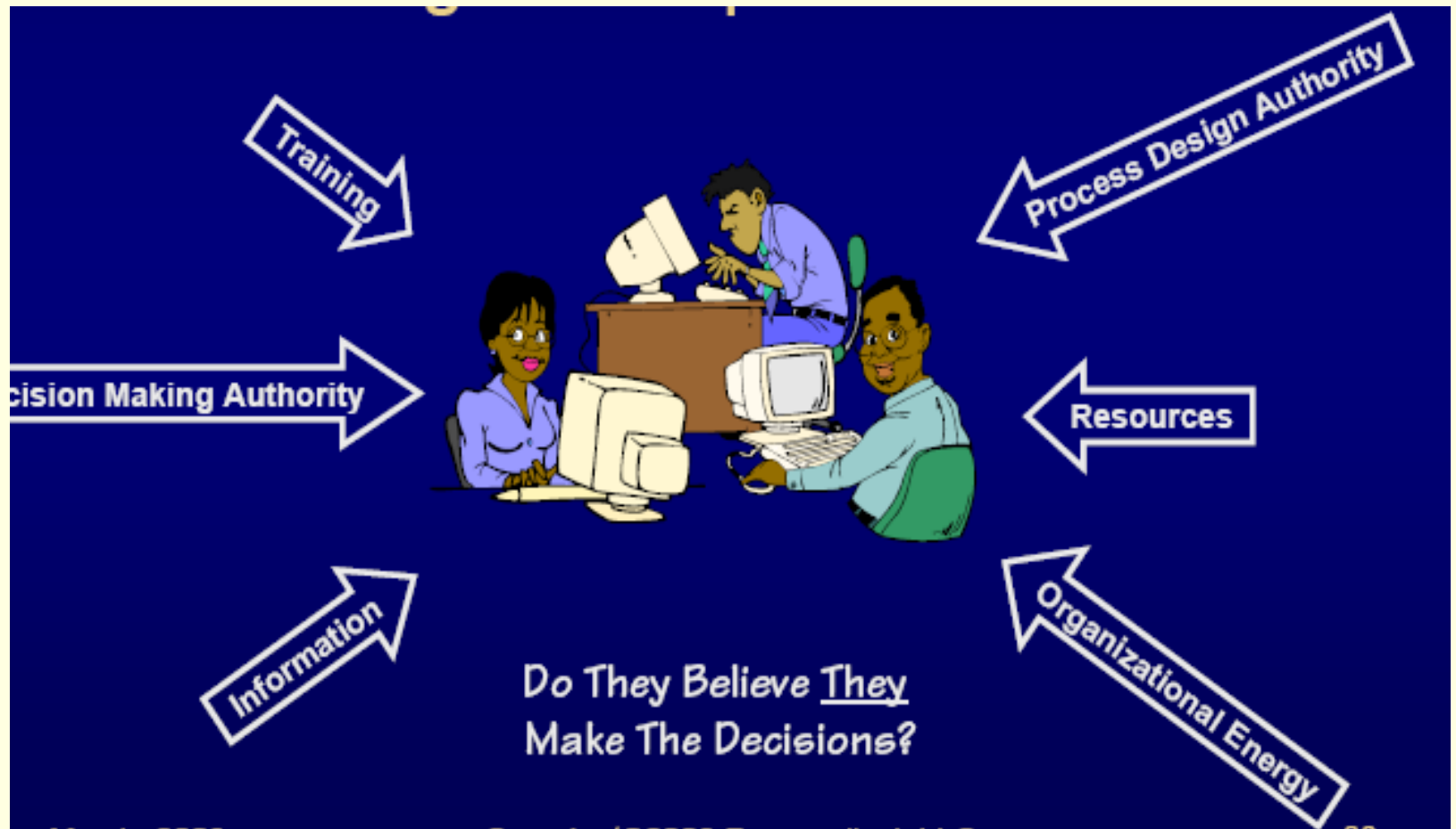


Principle #5: Empower the Team



- Does not mean to abandon leadership, but to let the people who add value use their full potential.
- While software development cannot be successful without disciplined, motivated people, experimentation and feedback are more effective than trying to getting things right the first time.
- The critical factor in motivation is not measurement, but empowerment: moving decisions to the lowest possible level in an organization while developing the capacity of those people to make decisions wisely.

Value those who add value



Empower the Team with..

➤ Motivation

- ❖ Intrinsic motivation requires a feeling of belonging, a feeling of safety, a sense of competence, and sense of progress

➤ Leadership

Managers	Leaders
<i>Cope with Complexity</i> <ul style="list-style-type: none">■ Plan and Budget■ Organize and Staff■ Track and Control	<i>Cope with Change</i> <ul style="list-style-type: none">■ Set Direction■ Align People■ Enable Motivation

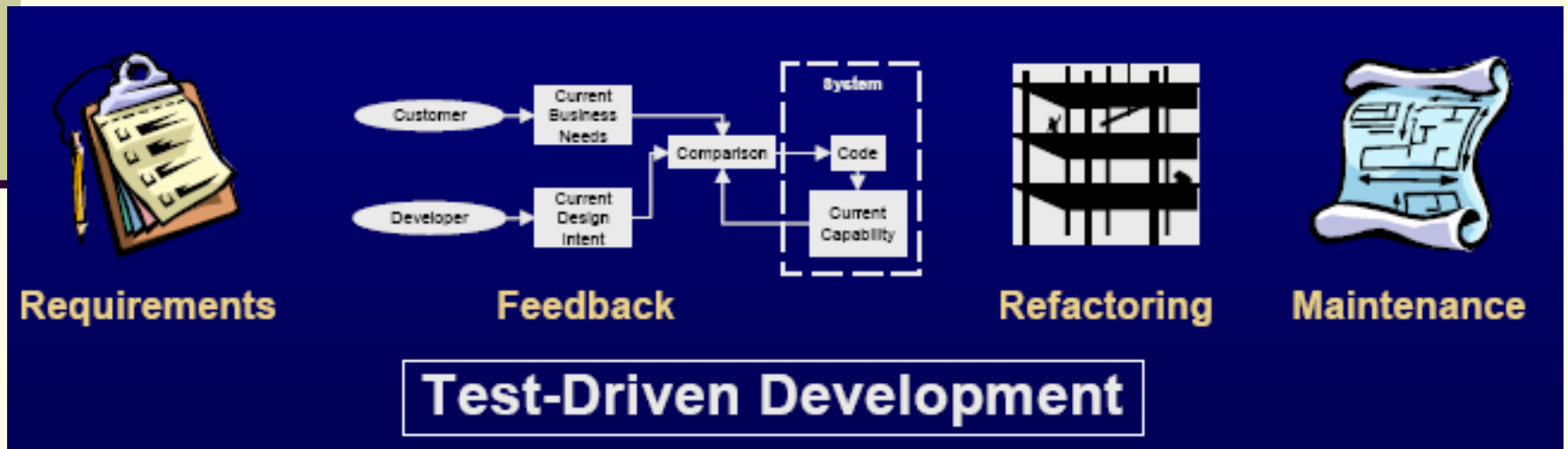
➤ Expertise

Principle #6: Build Integrity In

- Does not mean big, upfront design, but don't try to tack on integrity after the fact, build it in.
- **External (perceived) integrity** means that the totality of the product achieves a balance of function, usability, reliability, and economy that delights customers.
- **Internal (conceptual) integrity** means that the system's central concepts work together as a smooth, cohesive whole.
- The way to build a system with high perceived and conceptual integrity is to have excellent information flows both from customer to development team and between the upstream and downstream processes of the development team.

Build Integrity In With..

- Perceived Integrity
- Conceptual Integrity
- Refactoring
- Testing

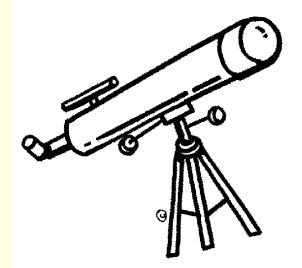


Principle #7: See the Whole



- Does not mean to ignore the details, but to beware of the temptation to optimize parts at the expense of the whole.
- Lean thinking suggests that optimizing individual parts almost always leads to sub-optimized overall system.
- Sub-optimization:
 - ❖ The more complex a system, the more temptation there is to divide it into parts and manage the parts locally.
 - ❖ Local management tends to create local measurements of performance. These local measurements often create system-wide effects that decrease overall performance.

See the Whole with..



➤ Measurements

- ❖ The way to be sure that everything is measured is by **aggregation**, not disaggregation. That is, move the measurement one level up, not one level down.
- ❖ **Information measurements** (obtained by aggregating data to hide individual performance), not performance measurements, should be used.

➤ Contracts

- ❖ Project managers have **four variables** that they can adjust when managing projects: **time, costs, quality, and scope**. From these four variables, fix time, cost and quality, but not scope. Prioritize features, but don't specify in the contract the fixed set of features to be delivered. Move from a fixed scope to a negotiable scope*: By delivering high priority features first, it is likely that you deliver most of the business value long before the customer's wish list is completed.

Conclusion

- The lean production metaphor is a good one for software development, if it is applied in keeping with the underlying spirit of lean thinking.
- The underlying principles of eliminating waste, empowering front line workers, responding immediately to customer requests, and optimizing across the value chain are fundamental to lean thinking
- When applied to software development, these concepts provide a broad framework for improving software development.



THANKS