Lean Software Development
Principles of Lean Thinking in Software Engineering
Lecture 8

Artem Kruglov, Giancarlo Succi

Innopolis University

a.kruglov@innopolis.ru, *g.succi@innopolis.ru*

10th February 2021

Principle of Lean Thinking in Software Engineering

Outline

- **Perspectives of Lean**
- Perspectives of Lean – Value
- Perspectives of Lean – Knowledge
- Perspectives of Lean – Improvement
- Push vs. Pull in Software Engineering

Perspectives of Lean

- **Value:** methods that support the organization to focus on the understanding and maximization of the delivered value;
- **Knowledge:** methods that focus on the creation of a shared understanding of the know-how, know-where, know-who, know-what, know-when, and know-why within the company;
- **Improvement:** methods that help to instill a culture of constant improvement.

Outline

- Perspectives of Lean
- **Perspectives of Lean – Value**
- Perspectives of Lean – Knowledge
- Perspectives of Lean – Improvement
- Push vs. Pull in Software Engineering

Value

Agility was developed to address software development risks:
trying to focus on processes that consist only of essential, value-adding activities.

Agility, on the other hand, was developed to address the intrinsic variability and randomness of software development.

In both cases the key idea is to focus on value, on activities that deliver **business value** to the customer

Managing Value

Value-driven in typical methods:

- DevOps
- Waterfall
- Spiral

Managing Value is always related to managing various types of risks in software production.

Risks in software production

Risks:

- the risk of not understanding the requirements of the customer;
- the risks that the customer changes his mind;
- the risks of choosing the wrong technology;
- the risks of writing defective software.

One way to quantify the risk is to calculate the risk exposure

Risk exposure =

probability of an unsatisfactory outcome $\times$

loss to the parties affected if the outcome is unsatisfactory

DevOps: development and operations

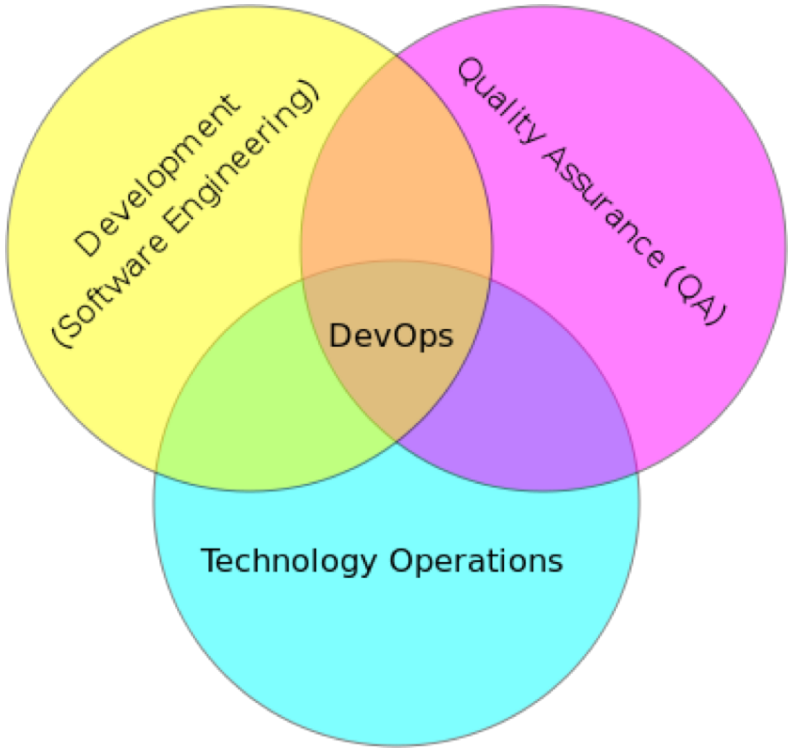
DevOps is a practice that emphasizes the collaboration and communication of both software developers and other information-technology. DevOps aims to close the gap between development and operations and integrates both side on three dimensions:

- Process integration
- Tool integration
- Data integration

[1] `https://en.wikipedia.org/wiki/DevOps`

DevOps: development and operations

Venn diagram showing DevOps as the intersection of development (software engineering), technology operations and quality assurance (QA)



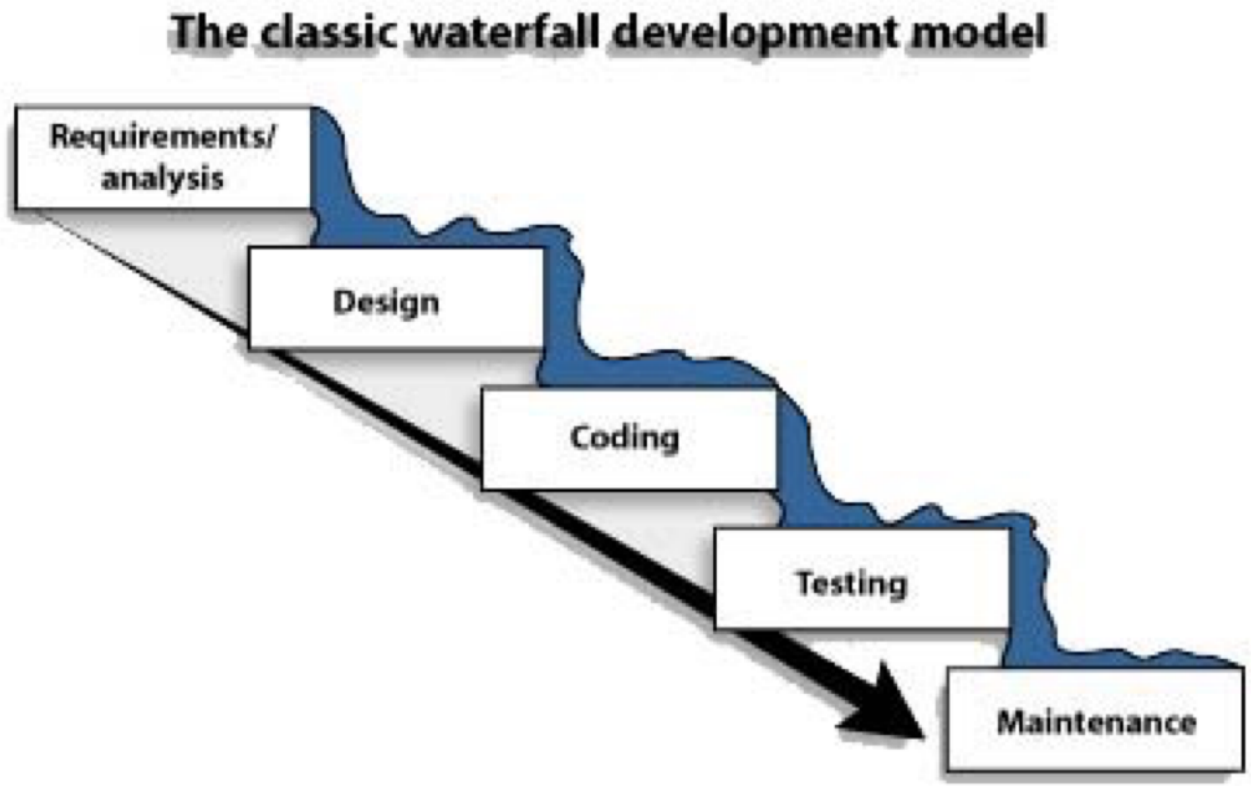
[1] https://en.wikipedia.org/wiki/DevOps

Waterfall development model

The waterfall model was the first to address risks, namely it focused on the to develop a product that does not correspond to the requirements. To handle such risk properly, the design was done only after the requirements were clear, the implementation was done after the design was completed, the system was then verified and tested at the end to again ensure that it did what it should do.

[1] http://www.railshorde.com/blog/waterfall-development-model

Waterfall development model – Schema



[1] http://www.railshorde.com/blog/waterfall-development-model

Waterfall development model

The phases in Waterfall model are discussed in detail as follow:

Planning and Requirements: All possible requirements of the system to be developed are gathered in this phase and planning for further development is done in advance to produce a Quality product within the given time and cost.

Analysis and System Design: The requirements captured in first phase are deeply analyzed and then a system design is prepared. This system design is the blue print of the system to be developed. System Design helps in defining hardware and software needs of the system and also specifies the overall system architecture.

[1] `http://www.railshorde.com/blog/waterfall-development-model`

Construction and Modeling: With the inputs of previous phase the actual coding is done in this phase to develop the system. The developer is responsible for the coding part. The system is first developed in small programs called units, which are integrated in the next phase.

Integration and Testing: All the individual units developed in the Construction phase are integrated into a system as a whole after each unit is tested thoroughly called as Unit testing. Then the whole system all together is tested for any faults or defects.

[1] `http://www.railshorde.com/blog/waterfall-development-model`

Waterfall development model

Deployment of system: After the Unit and System testing is done and all the bugs are resolved, the product is deployed to the customer.

Maintenance: There are some issues which are encountered by the client or the customer after deployment, fixing of those issues comes under Maintenance phase. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

[1] `http://www.railshorde.com/blog/waterfall-development-model`

Waterfall development model

Advantages of Waterfall Model:

- Simple and easy to use and understand.
- Each phase is processed one at a time, so full dedication is on one process only.
- Works well for smaller projects where requirements are well understood.
- Each stage has its own task to do deliberately.
- Each process is fully documented to avoid further difficulties.

[1] `http://www.railshorde.com/blog/waterfall-development-model`

Waterfall development model

Disadvantages of Waterfall Model:

- The biggest problem arises that no another process can come under work simultaneously.
- The requirements are to be predefined at the initial stage; if requirements are identified at later stage of time then they cannot be easily appended at the existing Waterfall Cycle.
- Waterfall Model does not allow iterations of phases. The whole project can be integrated at the end only.
- The customers can preview project at end only. No earlier prototype is released and shown to user.

[1] `http://www.railshorde.com/blog/waterfall-development-model`

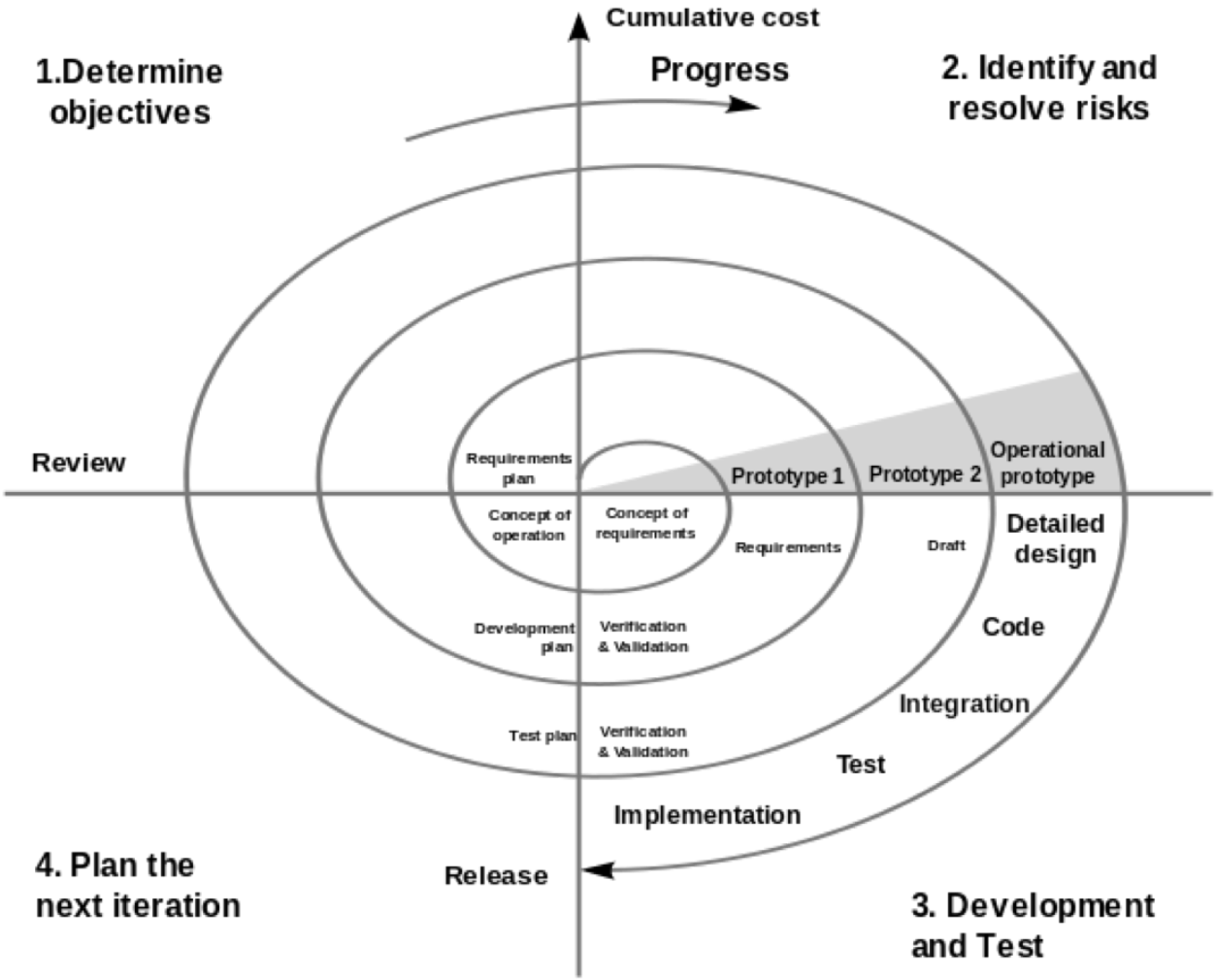
Waterfall development model

Applications of Waterfall Model:

- There are no ambiguous requirements.
- Requirements are very well documented and fixed beforehand.
- Technology is not dynamic.
- Product definition is stable.
- Sufficient resources and expertise are available to support the product.

[1] `http://www.railshorde.com/blog/waterfall-development-model`

Spiral development model



[1] https://en.wikipedia.org/wiki/Spiral_model

Spiral development model

The spiral software development model is an iterative software development approach in which each cycle shows the following characteristics:

- Concurrent rather than sequential determination of artifacts.
- Consideration in each spiral cycle of the main spiral elements:
  - critical stakeholder objectives and constraints,
  - product and process alternatives,
  - risk identification and resolution,
  - stakeholder review,
  - commitment to proceed.

Spiral development model

- Using risk considerations to determine the level of effort to be devoted to each activity within each spiral cycle.
- Managing stakeholder life cycle commitments.
- Emphasis on activities and artifacts for system and life cycle rather than for software and initial development.

Outline

- Perspectives of Lean
- Perspectives of Lean – Value
- **Perspectives of Lean – Knowledge**
- Perspectives of Lean – Improvement
- Push vs. Pull in Software Engineering

Knowledge

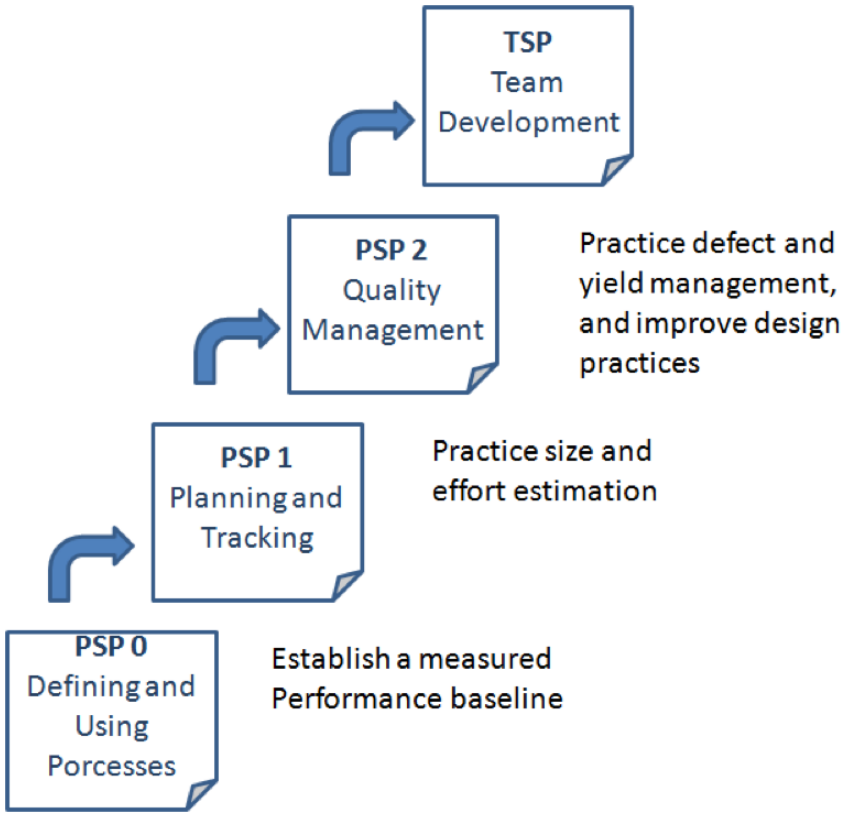
The problem of managing knowledge is strongly related to the problem of managing value.

Lean management in software focuses on methods that promote the creation of a shared understanding of the know-how, know-where, know-who, know-what, know-when, and know-why within the company. The collection, storage, organization, processing, and distribution of this understanding throughout the software organization are essential to deliver value.

- Personal Software Process
- Goal Question Metric paradigm
- The Balanced Scorecard approach

Personal Software Process (PSP)

An approach that collects data about what the single developer does and relates it to his performance is the Personal Software Process



The input to PSP is the requirements; requirements document is completed and delivered to the engineer.

Personal Software Process (PSP)

PSP0, PSP0.1 (Introduces process discipline and measurement)

PSP0 has 3 phases: planning, development (design, coding, test) and a post mortem. A baseline is established of current process measuring: time spent on programming, faults injected/removed, size of a program. In a post mortem, the engineer ensures all data for the projects has been properly recorded and analyzed.

PSP0.1 advances the process by adding a coding standard, a size measurement and the development of a personal process improvement plan (PIP). In the PIP, the engineer records ideas for improving his own process.

Personal Software Process (PSP)

PSP1, PSP1.1 (Introduces estimating and planning)

Based upon the baseline data collected in PSP0 and PSP0.1, the engineer estimates how large a new program will be and prepares a test report (PSP1).

Accumulated data from previous projects is used to estimate the total time. Each new project will record the actual time spent.

This information is used for task and schedule planning and estimation (PSP1.1).

Personal Software Process (PSP)

PSP2, PSP2.1 (Introduces quality management and design)

PSP2 adds two new phases: design review and code review. Defect prevention and removal of them are the focus at the PSP2.

Engineers learn to evaluate and improve their process by measuring how long tasks take and the number of defects they inject and remove in each phase of development. Engineers construct and use checklists for design and code reviews.

PSP2.1 introduces design specification and analysis techniques

Goal Question Metric paradigm

GQM is a methodology to specify measurement models; it provides a mechanism to define a measurement strategy, and it defines a measurement goal and the means to achieve this goal

GQM+ Strategies approach

GQM+ Strategies is an extension of the Goal Question Metric paradigm which "provides mechanisms for explicitly linking software measurement goals, to higher-level goals for the software organization,and further to goals and strategies at the level of the entire business

Goal Question Metric paradigm

Measurement helps us to quantify things we observe in the real world and is useful in three ways:

- Measurement helps to understand what is happening during the various activities in which programmers are involved; they make aspects of process and product more visible, giving a better understanding of relationships among activities and the entities they affect.

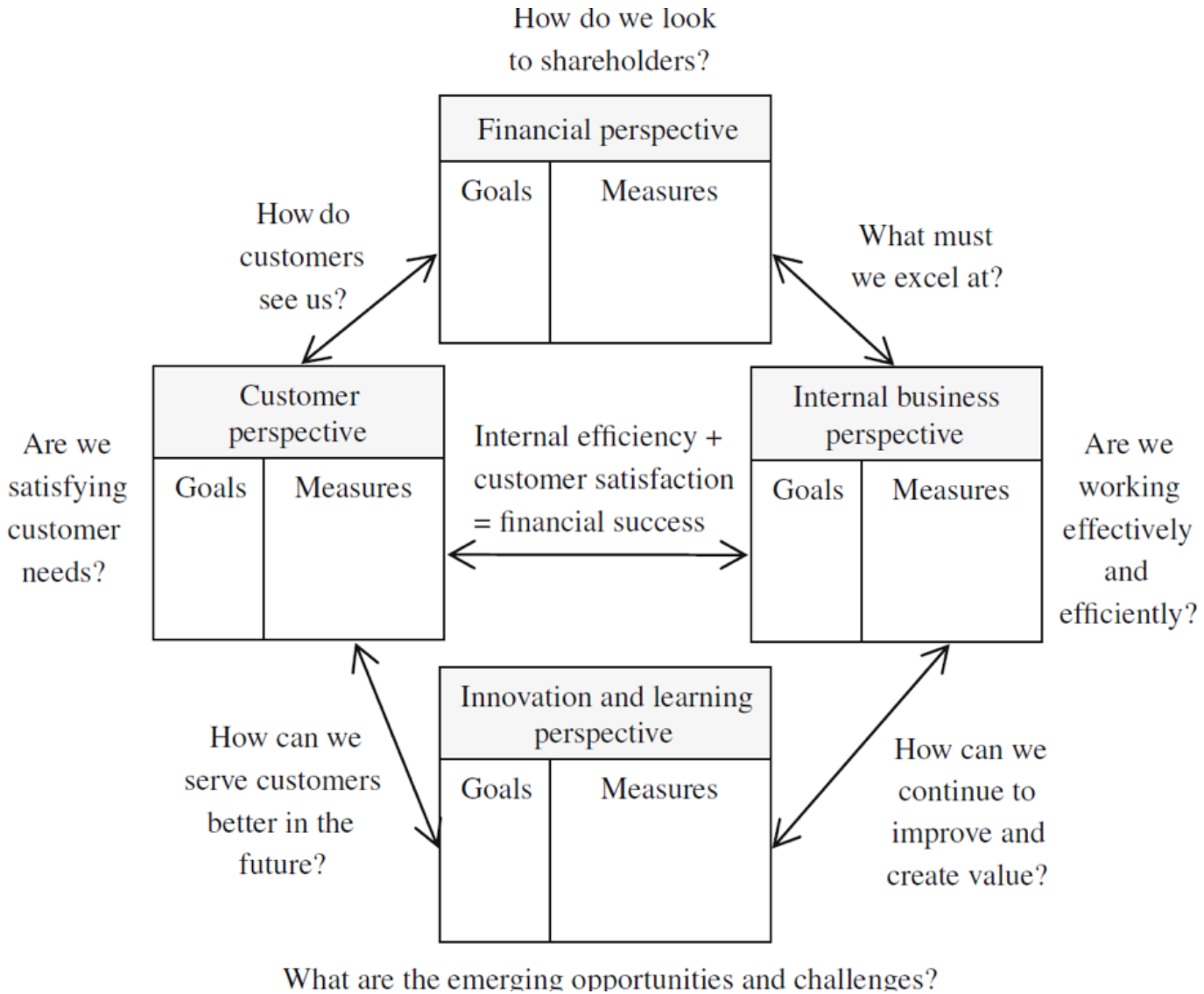
Goal Question Metric paradigm

- Measurement allows to control what is happening on our projects. The collected data are used to predict what is likely to happen and to make changes to processes and products that help us to meet our goals.
- Measurement helps to understand how to improve the processes and products. For instance, we may adopt a new software development technique, based on measures of its impact on the software quality.

The Balanced Scorecard approach

This is an approach to structure key performance indicators within an organization in form of scorecards and to focus on a set of scorecards that cover all parts of the organization to obtain a “balanced” picture.

The Balanced Scorecard approach



The Balanced Scorecard approach

Financial: assesses the organization from the shareholders' viewpoint: shareholder value, liquidity, revenue growth, productivity, resource utilization, efficiency, etc.;

Customer: assesses the organization from the customers' viewpoint: perceived quality of the product, reputation of the organization, customer loyalty, etc.;

The Balanced Scorecard approach

Innovation and learning: assesses the capability of the organization to innovate and continuously improve its processes looking at the available employee assets, information systems capabilities, and organizational infrastructure; and

Business process effectiveness: assesses the maturity of the business processes in efficiently and effectively providing the planned output.

Outline

- Perspectives of Lean
- Perspectives of Lean – Value
- Perspectives of Lean – Knowledge
- **Perspectives of Lean – Improvement**
- Push vs. Pull in Software Engineering

Improvement

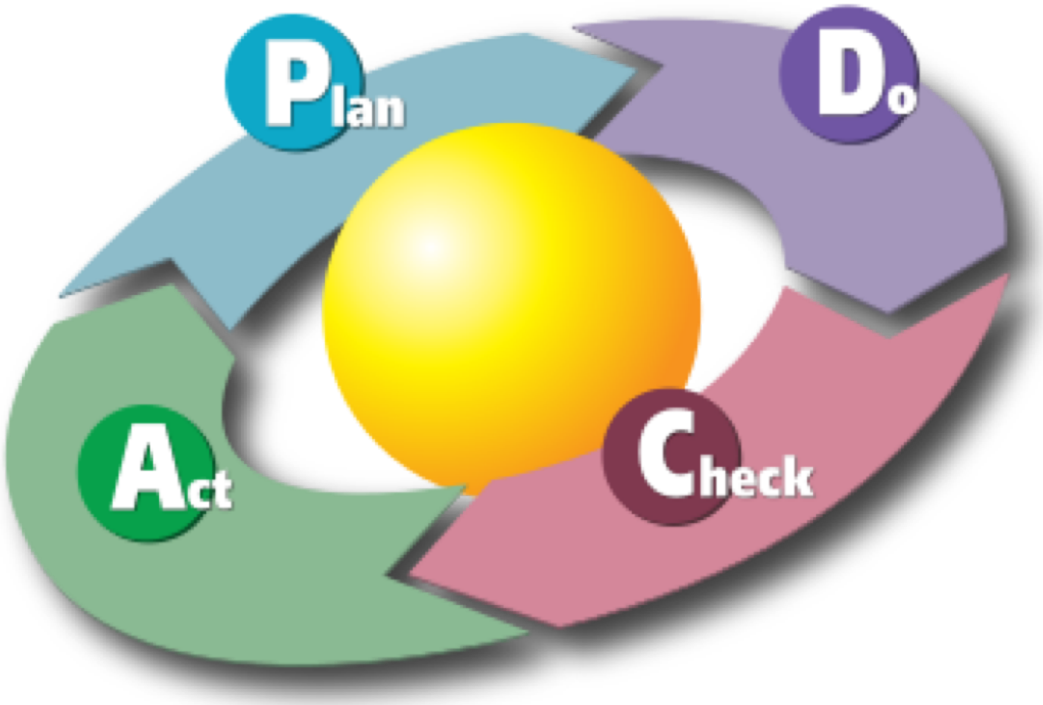
Constant improvement is also related to the problem of maximizing value:

it is essential to constantly verify if current working methods are still valid or if new technologies, new methods, past experiences, etc. can be used to improve.

- Shewhart cycle
- DMAIC cycle
- CMMI

Shewhart cycle

Shewhart cycle is an iterative four-step management method used in business for the control and continuous improvement of processes and products



[1] https://en.wikipedia.org/wiki/PDCA

Shewhart cycle

PLAN
Establish the objectives and processes necessary to deliver results in accordance with the expected output (the target or goals). By establishing output expectations, the completeness and accuracy of the specification is also a part of the targeted improvement. When possible start on a small scale to test possible effects.

DO
Implement the plan, execute the process, make the product. Collect data for charting and analysis in the following "CHECK" and "ACT" steps.

[1] `https://en.wikipedia.org/wiki/PDCA`

Shewhart cycle

CHECK Study the actual results (measured and collected in "DO" above) and compare against the expected results (targets or goals from the "PLAN") to ascertain any differences. Look for deviation in implementation from the plan and also look for the appropriateness and completeness of the plan to enable the execution.

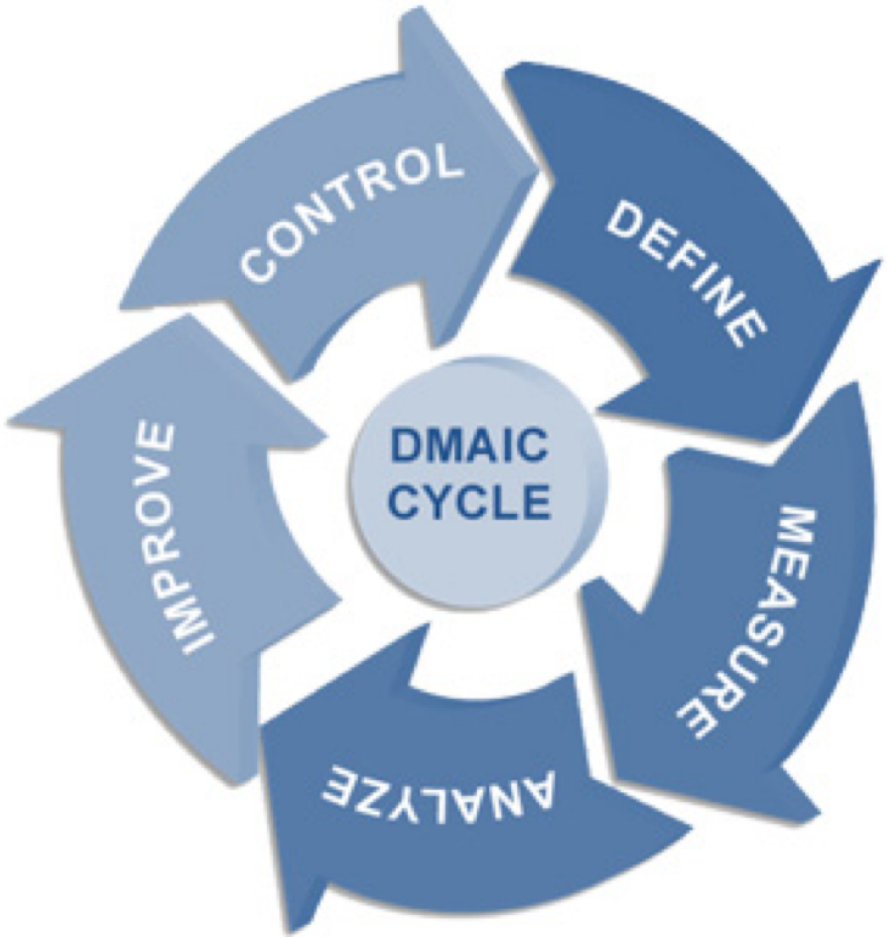
ACT If the CHECK shows that the PLAN is an improvement to the prior standard, then that becomes the new standard for how the organization should ACT going forward.

If the CHECK shows that the PLAN is not an improvement, then the existing standard will remain in place.

if the CHECK showed something different than expected, then there is some more learning to be done

DMAIC cycle

DMAIC cycle refers to a data-driven improvement cycle used for improving, optimizing and stabilizing business processes and designs.



[1] https://en.wikipedia.org/wiki/DMAIC

DMAIC cycle

Define

The purpose of this step is to clearly articulate the business problem, goal, potential resources, project scope and high-level project timeline.

Measure

The purpose of this step is to objectively establish current baselines as the basis for improvement. This is a data collection step, the purpose of which is to establish process performance baselines.

[1] https://en.wikipedia.org/wiki/DMAIC

DMAIC cycle

Analyze
The purpose of this step is to identify, validate and select root cause for elimination.

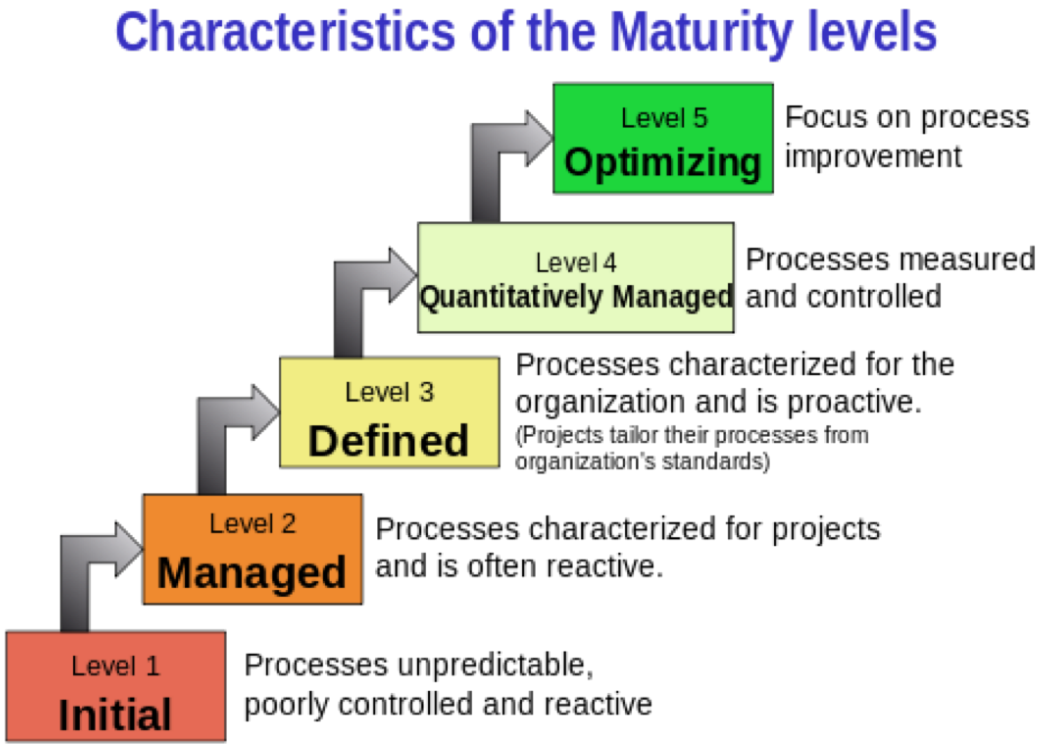
Improve
The purpose of this step is to identify, test and implement a solution to the problem; in part or in whole.

Control
The purpose of this step is to sustain the gains. Monitor the improvements to ensure continued and sustainable success. Create a control plan. Update documents, business process and training records as required.

[1] `https://en.wikipedia.org/wiki/DMAIC`

Capability Maturity Model Integrated CMMI

The CMMI is a family of reference models covering the development and maintenance activities applied to both products and services.



[1] https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration

Outline

- Perspectives of Lean
- Perspectives of Lean – Value
- Perspectives of Lean – Knowledge
- Perspectives of Lean – Improvement
- **Push vs. Pull in Software Engineering**

Push vs. Pull in Software Engineering

Requirements-First Development

The "push" approach is present whenever activities are triggered as soon as other activities finish their work, obtaining their output as input.

To begin the software development process, collecting requirements is practiced in nearly all software development approaches.

The definition of requirements converts business objectives into requirements that take into consideration the technical possibilities and their costs. This step requires considering the benefits and the costs of a technological solution in solving a business problem.

Push vs. Pull in Software Engineering

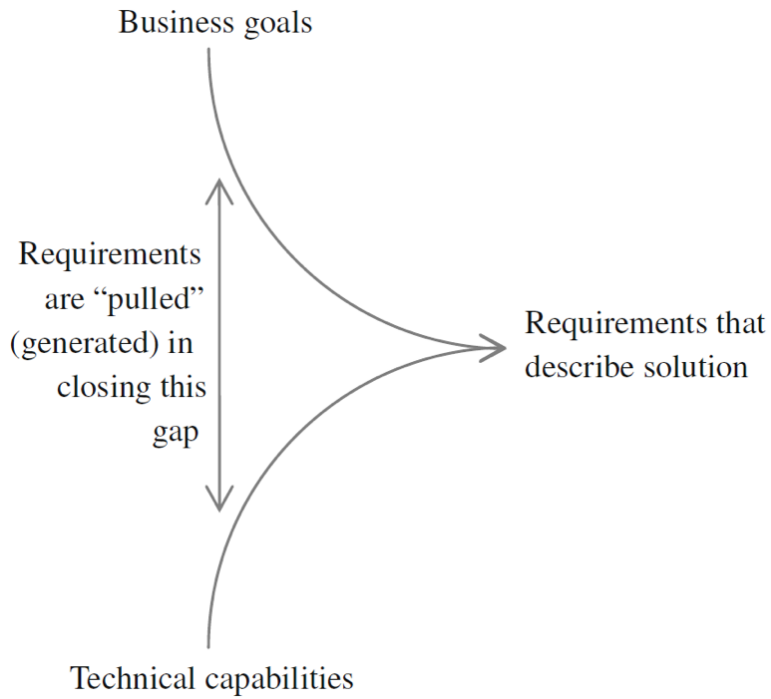
Requirements-First Development

This causes costs related to the following situations:

- the requirements do/do not optimally contribute to fulfill the business objectives;
- requirements change because old requirements have shown not to contribute to the fulfillment of the business objectives;
- the implementation of the requirements costs more than an alternative implementation that fulfills the business objectives in the same way;
- requirements that would contribute to the business objectives and that would not be costly to implement are not requested.

Requirements-First Development

A possible "pull" approach can be achieved through an approach in which goals are collaboratively identified before setting the requirements and used to "pull" requirements and their priorities from the stakeholders



Easy Win-Win (1/2)

A Groupware-Supported Methodology For Requirements Negotiation

EasyWinWin is a requirements definition methodology that builds on the win-win negotiation approach and leverages collaborative technology to improve the involvement and interaction of key stakeholders.
This methodology includes the following steps:

- Step0. Engage the success-critical stakeholders.
- Step1. Refine and Expand Negotiation Topics.
- Step2. Brainstorm Stakeholder Win Conditions.
- Step3. Brainstorm Stakeholder interests.
- Step4. Define a Glossary of Key Terms.

Easy Win-Win (2/2)

A Groupware-Supported Methodology For Requirements Negotiation

- Step5. Prioritize Win Conditions.
- Step6. Surface Issues and Constraints.
- Step7. The WinWin Tree: Win Conditions, Issues, Options, Agreements.
- Step8. Organize Negotiation Results.

Source: `http://csse.usc.edu/csse/research/easy_win_win/` and `http://www.computer.org/csdl/proceedings/hicss/2002/1435/01/14350021b.pdf`

Bottom-Up Development

In a bottom-up approach, the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed.

Building blocks are an example of bottom-up design because the parts are first created and then assembled without regard to how the parts will work in the assembly.

[1] https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design

Top-Down Development

Top-down is a programming style, the mainstay of traditional procedural languages, in which design begins by specifying complex pieces and then dividing them into successively smaller pieces.

The technique for writing a program using top-down methods is to write a main procedure that names all the major functions it will need.

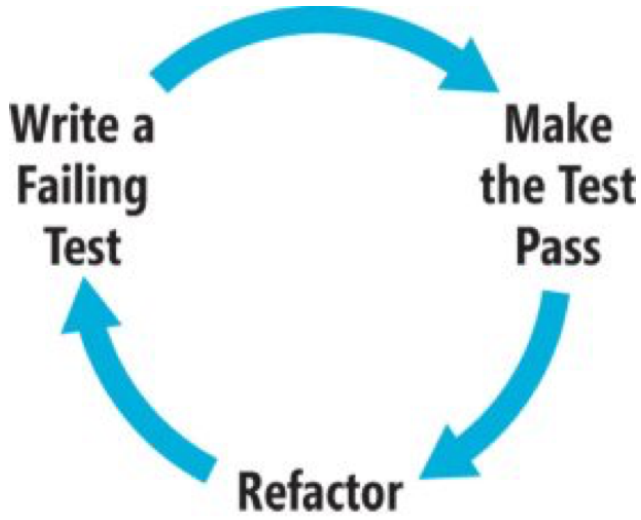
Later, the programming team looks at the requirements of each of those functions and the process is repeated.

[1] https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design

Test Driven Development

Test-driven development(TDD) is a software development process that relies on the repetition of a very short development cycle:

- The programmer writes a test case that tests the desired functionality as if it were already present in the code.
- The initial test case fails, but it describes the expected outcome of the new capability.
- The next step is to make the test pass implementing the necessary code.
- Refactor and pick the next requirement (go back to step 1).



Summary

Year	Proposal	Value	Knowledge	Improvement
1970s	Rapid application development (RAD) [35]	×		
1985	Quality improvement paradigm [6]			×
1986	Scrum (product development) [49]	×		
1986	Spiral software development [13, 15]	×		
1987	Six sigma [39]			×
1987	Goal question metric approach [8]		×	×
1989	Experience factory [7]		×	×
1991	Capability maturity model (CMM) [40]			×
1992	Balanced scorecard [33]		×	
1995	Dynamic systems development method (DSDM) [48]	×		
1999	Extreme programming (XP) [10]	×		
1999	Feature driven development [19]	×		
2000	Adaptive software development [28]	×		
2000	Personal software process [30]		×	×
2001	Agile manifesto [11]	×		
2002	Capability maturity model integration (CMMI) [18]			×
2004	Scrum (software development) [44]	×		
2004	Crystal clear [20]	×		
2007	GQM$^+$ strategies [9]		×	×
2009	DevOps [31]	×		
2011	Disciplined agile delivery [4]	×		