USING LEAN PRINCIPLES AND MBE IN DESIGN AND DEVELOPMENT OF AVIONICS EQUIPMENT AT ROCKWELL COLLINS

David Lempia
Principal Systems Engineer
Rockwell Collins, Inc.

Keywords: *Modeling, Design, Development, Lean, Value Stream*

Abstract

For over 75 years, Rockwell Collins has been recognized as a leader in the design, production, and support of communication and aviation electronics for customers worldwide. Today at Rockwell Collins, modeling technologies and Lean Principles are brought together for the design and development of complex safety critical systems. Rockwell Collins calls this the Model-Based Engineering (MBe) initiative.

The Model-Based Engineering (MBe) vision is to increase shareholder value and customer satisfaction by reducing product development cost and cycle time using Model-Based Engineering enabled Lean Engineering principles. The goal of MBe is to achieve the same results in the design and development process that Lean has achieved in operations and manufacturing. In other words, the goal is to create a Lean design and development factory.

This paper describes the Lean principles enabled by MBe technologies and presents the performance improvements resulting from using MBe technologies and Lean Principles for complex avionics systems designs at Rockwell Collins

1 Introduction

What is Model Based Engineering (MBe)? MBe is the systematic use of models as primary engineering artifacts throughout the product development lifecycle. The emphasis of MBe is actor centric refinement of requirements and model centric development needed to:

1. validate customer needs that are expressed in the model,
2. achieve behavior and performance requirements expressed in the model,
3. generate lifecycle artifacts from the model, and
4. verify that the design model is implemented in the product without error.

Examples of artifacts that can be generated from a model include blueprints, software, interface configurations, and test vectors.

2 Lean Principles

What is Lean? Lean is the relentless elimination of waste to increase value as defined by the customer. Lean provides more value to customers by utilizing tools, methods, and principles to:

1. eliminate waste throughout the value stream,
2. create processes that are adequate, available, capable, flexible and value added, and
3. continuously drive toward perfection.

The principles of Lean are value, value stream, pull, flow, and perfection as depicted in Fig. 1. [1]

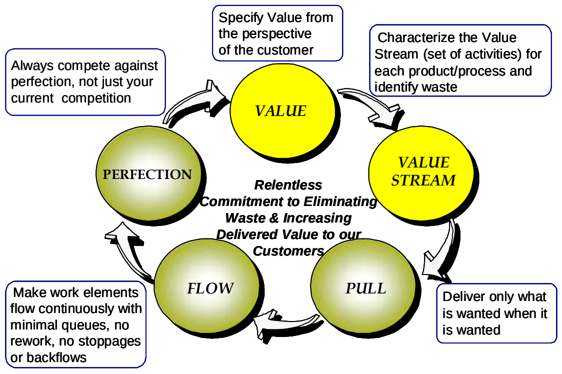
Specify Value from the perspective of the customer
Characterize the Value Stream (set of activities) for each product/process and identify waste
Always compete against perfection, not just your current competition
VALUE
VALUE STREAM
PERFECTION
Relentless Commitment to Eliminating Waste & Increasing Delivered Value to our Customers
FLOW
PULL
Make work elements flow continuously with minimal queues, no rework, no stoppages or backflows
Deliver only what is wanted when it is wanted

Fig. 1. Lean Principles

2.1 Define Value from the Customer Perspective

The first Lean principle is to specify value from the perspective of the customer. Lean is the focus of each and every employee on a team. Lean is the driving force behind the decisions teams make in selecting tools, process, and training.

To be effective in defining value, teams need to involve both internal and external customers. They should create a strong working relationship with the customer to understand their culture, value system, approach, attitude, expectations, and issues. They need to validate the customer needs often using tools such as mockups, prototypes, and design documents. They need to be willing to challenge the customer's assumptions and to clarify the requirements. Finally, they need to communicate the customer needs throughout the program. In summary, teams need to promote a culture of putting the customer first. This is depicted in Fig. 2.

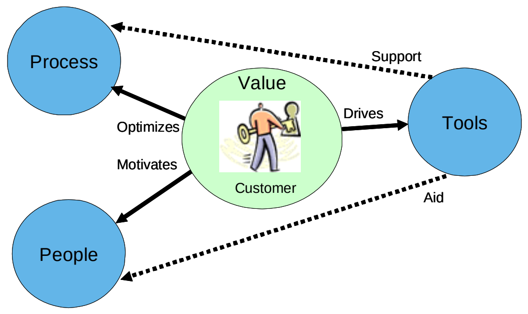

Fig. 2. Customer Defined Value

Because the customer is the focus, teams can use customer-defined value as an assessment tool to help optimize the process, to select and define tools, and to focus and train people. Tools, such as MBe tools, must both help the process and help people provide value for the end customer.

2.2 Identify the value stream.

The second Lean principle is to identify, develop, and utilize a value stream. The value stream is the set of specific actions needed to bring a specific product through the 3 critical management tasks for any business.

1. Problem Solving Task. This is the set of problem solving tasks between concept inception to the development of talent and intellectual property to the final production launch.
2. Information Management. This is the set of information management tasks from customer order to customer delivery.
3. The Physical Transformation. This is the set of build steps from raw material and customer requirements to product in the hand of the customer.

The design and development team impacts each of these value streams. Team members need to understand their value streams and how they affect the value streams of other teams. The value stream helps team members understand who their customers are, map value-added activities as defined by the customer, and identify activities that add little or no value to the customer.

Effective value streams are developed and documented using a cross functional team consisting of the product development stakeholders. There are various aids to help create value streams such as the SIPOC (Suppliers Inputs Process Outputs Customers) diagram [2], data boxes to capture activity details and technical or social issues, and process flow diagrams. Value stream maps are created for the current state, a future state, and the ideal state. The evolution of the tools, processes, and training are methodically worked by a team to move from the current value stream to the future value stream and eventually to the ideal value stream using short term and long term action plans.

Design and development teams start a project using a current-state value stream map. Over the course of the project they mature the tools, processes, and culture to a future-state value stream map as defined in the short term action plan. Likewise, groups of projects collaborate at the enterprise level to mature the collective value streams beyond the future state by implementing additional ideal state characteristics using the long term action plans.

The six steps in creating a value stream include:

1. Identify the process steps in order, beginning and ending with the customer.
2. Complete data boxes for each process step.
3. Map the information flow, designate what is being communicated and who is giving/receiving it.
4. Identify what information is flowing in & out of each process step.
5. Calculate the Takt and Cycle time.
6. Analyze your process and identify waste.

Takt Time is a measurement of the rate of the available time divided by the unit demand. For example, if 40 units need to be built over the course of 80 available hours, the Takt Time is 2 hours per unit. For every 2 available hours of work, one unit needs to be built. The Takt time spreads the customer demand evenly across available time. A unit should be developed at the end of each Takt time. This enables predictable and linear flow. Cycle time is the time it takes to perform each activity. Takt time and Cycle time can be balanced to help level load a process. [1]

The last step in the value stream is to analyze the process and identify waste. There are 7 types of waste. [3]

1. Transportation – Moving material or information.
2. Inventory - Having more material or information than the customer wants.
3. Motion - Moving people to access or process material or information.
4. Waiting- Idle delays where you can't do your job.
5. Over-production – (Making too much) Making more material or information than required by the customer.
6. Over-processing - Effort and time spent that adds no value.
7. Defects - Errors or mistakes causing the effort to be redone.

These forms of waste are depicted in Fig. 3.

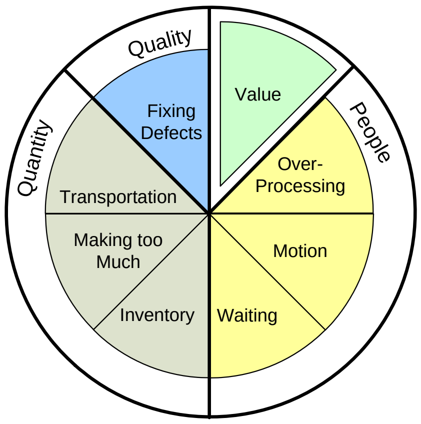

Fig. 3. Types of Waste

After the seven forms of waste are removed, customer defined value remains.

2.3 Enable pull system.

In a pull system, product is pulled at the rate demanded by end customers. The opposite of a pull system is a push system. A push system is the way project teams form naturally to produce products. In a push system, the development runs as fast as it is able to and creates as much product as possible. Inventory builds up at bottlenecks in the process. When items do not arrive in time people are assigned to find them and to expedite their movement through the system. The system rewards people for fixing problems. In a push system, the organization resembles the Wild West. Cowboys become heroes and are rewarded for "riding up on their stallion and saving the day".

In a pull system, cowboys are not needed. Units are not produced until the customer of the process pulls or requests delivery. The value stream maps each activity and the time it takes for it to be completed. The Takt time is used to lock-step the activities and to synchronize these activities with customer demand. The pull is initiated with adequate time to complete the activity. Because items are not produced until they are needed, inventory diminishes. People are not required to expedite products through the system. The cowboy fades off into the sunset. A pull system eliminates all seven forms of waste.

It is easy to visualize how this can happen in a factory that produces a physical product. In section 4 the Takt time is translated from the factory to the design and development paradigm.

2.4 Create smooth value stream flow.

A smooth value stream processes information (adding value) one piece at a time without interruption. A smooth value stream will not occur if there is waste of inventory, waiting, or defects. Process variability results in inventory and waiting. Defects result in re-work which results in inventory and waiting.

A smooth value stream is synchronized around the Takt time. Activities with cycle times that are the same as the Takt time can be lined up and performed in a smooth flow.

Standardized work, work sequence, work load leveling, and error-proofing are four mechanisms that help create a smooth flow of work when the current activity cycle times do not match the Takt time.

Standardized work identifies and documents repeatable activities. The standardized work is the best, easiest, and safest way to perform an activity. It is a documented set of work instructions arranged in a work sequence. The time it takes to perform standardized work is the cycle time.

Load leveling is arranging the activities and the resources such that the Takt time is achieved at each step in the process. For example if one activity takes twice as much time as the Takt time, arranging two resources in parallel will help the process flow smoothly.

Defects interrupt the smooth flow. Each defect that escapes from one activity and is caught in the next activity must be re-run. Re-running product through steps in the process results in the downstream step waiting for a product and the upstream steps queuing extra defective product. The result is an increase in inventory and a decrease in production. Error-proofing is designing standardized work and tools that prevent errors and results in a smooth flow of product.

In a load-balanced process each step has adequate time to finish before the next Tact time. The finished process steps create a natural pull through the system.

2.5 Work to perfection.

Perfection is a vision, not a goal. Striving for perfection will ensure a culture of continuous improvement. A tool for continuous improvement is the value stream and the action plan. A current, future, and ideal value stream become the roadmap that focuses the continuous improvement activities. The short term and long term action plans drive change from the current to the future to the ideal value stream.

The short term action plan identifies the changes that a project team can take to eliminate waste that is immediately avoidable or avoidable in a short period of time. When the short term action plan is complete, the future state value stream becomes the current state value stream.

The long term action plan identifies the changes that an enterprise group or set of projects needs to coordinate to eliminate waste. Long term projects invest in continuous improvements that eliminate waste for multiple teams. The ideal state characteristics drive the improvement to the existing current and future value streams.

The project teams and the enterprise group periodically re-evaluate and update the future and ideal value stream as they continue to strive toward perfection.

3 Model-Based Engineering

Lean Engineering defines principles that drive the culture of project teams. MBe is a technology improvement that eliminates waste from the value stream. To understand how MBe eliminates waste, it is important to understand models, modeling, modeling languages, frameworks, and simulation.

3.1 Model Basics

A model is the primary engineering artifact behind MBe. The model is not the product that is delivered to the customer; it only represents some aspect of the product. Numerous models represent the final product. For example, Fig. 4 shows a representation of the product's structure or system.

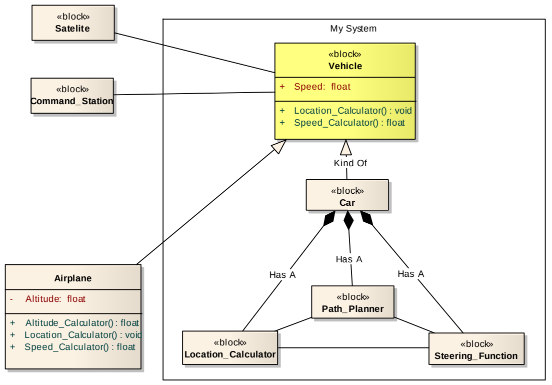

Fig. 4. Systems Model Example

Fig. 5 is a view of a model that animates manufacturing process used to assemble the parts defined by the system model.

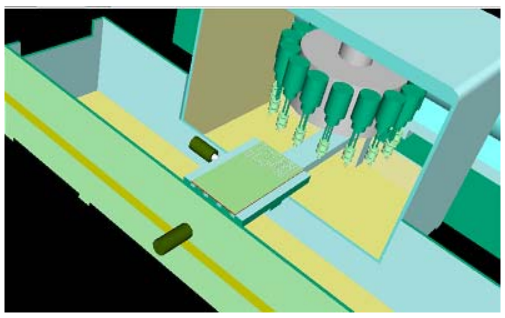

Fig. 5. Manufacturing Model Example

Finally, Fig. 6 is a model of the software used to generate the behaviors for one of the parts in the system model of the product.

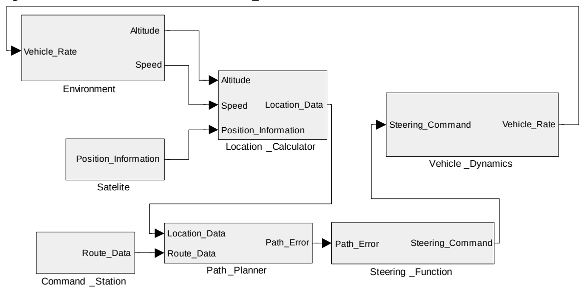

Fig. 6. Software Model Example

Each of these three example models represents a work product. The application of a standard structured methodology to create and validate a model is called modeling.

Models are developed using modeling languages. These languages often have a graphical and textual representation. Modeling languages express information, knowledge or systems and are defined by a consistent set of rules.

Examples of graphical design languages include SYSML (System Modeling Language) [4], UML (Unified Modeling Language) [5], AADL (Architecture Analysis Definition Language) [6], MARTE (Modeling and Analysis of Real-time and Embedded Systems) [7], and IDEF0 (Integrated DEFinition Function Modeling Method) [8], State Charts, FFBD (functional flow block diagrams), and control flow diagrams.

Each different design language is good at modeling one part of a system. For example, a FFBD is good at describing data flows from a producing function to a consuming function. The FFBD is not good at depicting the hierarchical structure of a system. Likewise, a schematic capture tool is very good at capturing the electrical layout of parts on a circuit card but not effective at capturing the functional data flow. Design languages are generally specialized to specific domains.

Models can be simulated. Simulation is the act of imitating the behavior of a system or process by stimulating the model. Simulation enables testing and performance optimization. Results of a simulation can be presented as the simulation runs or after the simulation finishes. These results are often depicted with graphs, plots, and user interfaces. Simulation gives the developer feedback of the behaviors of a model from the perspective of the end user. Simulations that animate the model give the developer a cause and effect diagnostic tool. For example, if the visible effect of a simulation scenario causes the end users display to jump inappropriately, the cause might be a filter that was not initialized at the right time. To find this, the model can be executed one step at a time. The effect of the execution of each block is examined on the user display. When the problem filter block is executed, the display jumps. The designer can then look at scenario leading up to the problem, visually see the cause-effect relationship, and use this information to develop a solution. An example showing the model of an airplane and the resulting time-history plot of an aircraft maneuver is shown in Fig. 7:

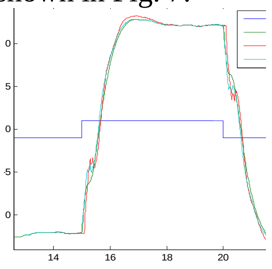

Fig. 7. Simulation Example

Traditional development processes create a requirements document and a design document. Based on the information in these two artifacts, the software is written. These two documents and the software all represent an aspect of the final product and can be considered a type of model. However, these are not the types of models that MBe needs to eliminate waste and reduce cycle time and cost. What characteristics should models have and what types of automation should tools support to help eliminate waste?

3.2 Models that Eliminate Waste

To help eliminate waste, models need enough rigor and formalism to enable tool automation that eliminates waste. In general, tool automation does two things. First it eliminates or identifies errors as models are created, analyzed, and simulated. Second it seamlessly or automatically moves and transforms data from one model to another model or from a set of models to the artifacts used in the final product.

Errors may be introduced as models are created. Construction errors can be identified using editors that understand the assembly rules of the modeling language. Model construction errors can be reduced or eliminated by using smart editors that check the assembly rules of the modeling language.

Once the models are created, they may not exhibit all of the required properties called out in the requirements. Models can be analyzed using model checkers to see if properties always hold true or never hold true. These models may also have hidden, missing, or incorrect behaviors designed in. Behavior can be simulated in executable models. The result of the simulation is animated in the editor or is depicted using time-history plots or other visualization techniques. Each of these model-enabled capabilities validate that the model is behaving correctly.

Errors may also be introduced as the models are translated for use in the final product. To convert the model into the final product, translators or generators are used to create design artifacts such as software and executables. The process of translating the model to the final product can inject errors. Qualified tools are developed and tested to a level that eliminates the translation errors. Artifact generators such as software generators, document generators, test generators, model viewers, and simulators all may be qualified. Qualified tools reduce the number of manual steps that are required for developing safety critical software under regulating process guidance such as DO-178B. [9]

4 Themes of Change

Change is difficult and painful for individuals and for organizations. The customer defined value and the value stream is a way to focus teams on solving the right issues. Value streams can be smoothed and waste can be eliminated by introducing new technology (tools) such as MBe. To minimize waste using MBe, the benefits and shortcomings of MBe must be understood. The proper application of MBe using Lean principles can radically reduce the waste in design and development processes.

To help with culture changes, six communication themes are described in this section. These themes simplify and depict the key characteristics of Lean and maximize the benefits of MBe in a Lean process. These themes are called “Customers Define Value”, “Frontload Design”, “Pull-a-little”, “Test-a-little Build-a-little”, “Single source design data”, and “Good Enough Common Solutions”. These themes of change need to be taught, mentored, and re-taught to new teams starting to apply Lean MBe at any company.

4.1 “Customers Define Value”

The foundational theme is also the first Lean principle as described in section 2.1. In summary of this section, customers define what value is and what value is not. All of the remaining themes apply in the context of customer-defined value and focus on creating smooth value streams.

Models of customers’ needs can be created using MBe techniques to help understand needs and transform the customer needs into requirements. Models provide both a way to assess the impact of changes and the ability to quickly prototype the change when the customer needs and requirements change.

4.2 “Frontload Design”

In traditional development experts pick a design based upon their expertise. Developers interpret what the design is. As design problems are uncovered, band aids are added to the design. Changes occur often and are not communicated. Developers do not understand the design decisions and are not committed to the design. The design is a shifting target.

Frontload Design encourages cross functional teams to carefully balance the different customer or stakeholder needs and to create multiple design options early. The team quickly eliminates the options that will fail (the fail fast principle). The good designs are refined and the process repeats itself until the best design emerges. Because the final solution has weathered the test of time, fewer design changes occur.

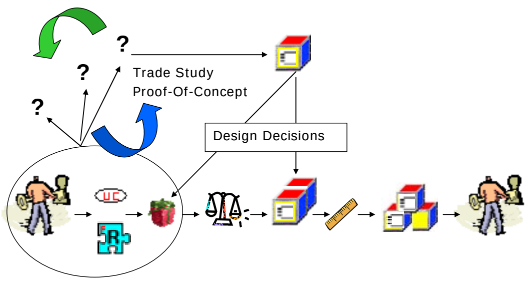

Fig. 8. Frontload Design Process

In Fig. 8, the frontload design process is summarized. The stakeholders (person holding a key) work with team members to describe their needs in the form of use cases (UC) and requirements (puzzle piece). The stakeholders include a broad set of interested parties such as end customer(s), users, and government regulators. Key technology risks are identified and used to identify the work units (the wrapped present). The work units are the tasks team members work on each day. For each of the key technology risks, numerous design options are captured and described (the question marks). Each design option is studied and understood using techniques as simple as white-board prototypes. After the numerous design options are studied, the team down-selects the options to the most promising designs. The promising designs are further refined (the arrows) using more complicated techniques such as aircraft flight deck simulation proof-of-concept. Trade-studies are conducted to select the best design option (the box). The selected design solution is documented and validated with key stakeholders. The design and development teams work in the framework of this design as they implement each use case and the associated requirements.

4.3 "Pull-a-little"

The "Pull-a-little" theme is pull as described in section 2.3. The word pull was renamed to Pull-a-little to emphasize pull and small tasks. This section refines and applies the concept of pull from its original factory application to this design and development environment application.

Traditional design and development process collect large quantities of requirements, use cases, high-level design, and test documents before moving to the next step in the process. Pull-a-little encourages engineers to identify small well-defined work units and to pull them all the way through the design and development process. Each work unit is created and made available at the Takt time. When an engineer finishes one work unit they pull the next work unit.

The concept of Takt time originates in a factory that produces units at a specific rate. The production rate is set to match the customer demand rate. What is the customer demand rate for design and development? The customer demands product availability at a specific date with a specific set of features or capabilities. Each feature takes a different amount of time to implement.

To translate a date and a list of features into a Takt time, a work unit and a design cycle need to be defined. A work unit is a task defined by a team and estimated to take one average person one working day to accomplish. The formula for Takt time is:

$$T = H / W \qquad (1)$$

where T is the Takt Time, H is the number of hours of available in a work day, and W is the number of work units that need to be completed in one day (by definition, this is also the number of people on the project).

A design cycle looks forward for one month of work days (~ 22 days). The team defines the work units for the entire design cycle in the beginning of a design cycle. To do this, the team starts with design decisions and a prioritized list of use cases and requirements. They then develop a list of tasks. Finally they refine the task list until each task is estimated to take one day to implement. At the beginning of each day, team members share any issues they have. Issues are assigned to the appropriate person (internal to the team or external to a support organization). Progress metrics are updated. Team members then select and implement the next work unit. The design cycle ends with an acceptance test and delivery of a working set of use cases.

Larger design teams may consist of numerous small teams working to a common architecture. A technical and program management team resolves issues and identifies coordinating design decisions, and creates prioritized use cases and requirements for each team. The teams are re-synchronized at the start of each design cycle. The deliveries from each team are integrated together and tested as part of the next design cycle. Design cycles end and support begins with the delivery of the design for manufacturing.

4.4 "Test-a-little Build-a-little"

Traditional process and especially model-centric process teams will develop a design (model) first and then develop tests after the function is working. The design is often developed as a large batch with a mix of new functions that work and have been tested, functions that may work but have not been tested, and functions in various stages of development. Testing is pushed to the end. Design errors caught in testing at the very end may drive large changes and result in rework and schedule slips.

Test-a-little Build-a-little encourages engineers to create a measure of success (test) for a small set of functions (work unit) from the eyes of the customer and then to build only what is needed in the model to pass this test. Using the inputs from upstream customers and standardized communication languages and building blocks, the models are developed one piece at a time. The work is designed to pass the test. Any model parts not needed to pass the test are waste and should be eliminated. For example, developing place holders for future functionality is waste. It is better to re-factor an existing design to fit a new function. Small fully-tested functions are created and passed on for integration. Engineers find defects early in the design process and reduce waiting, over-production, and inventory.

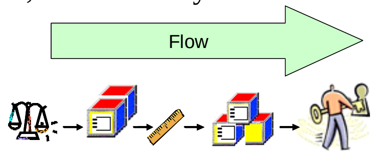

Fig. 9. "Test-a-little Build-a little" Process

In Fig. 9, the measure of success (Scale) is defined, the design for one work unit is developed (Box) until the test (Ruler) passes. The new function is integrated (pyramid of boxes) every design cycle and tested for customer acceptance (Person).

Test-a-little Build-a-little adds quality at every step in the process or error-proofs the process. A process that is error-proof has less variation and improves the smooth flow of design development through the value stream.

4.5 "Single Source Data"

Traditional processes are document centric and find engineers re-entering design information multiple times in the value stream. Each new entry of data must be maintained and updated to be correct. If a bug is found in the design, engineers often look at the software first even if the design is easier to understand. This is because the software is closer to the truth about what the system does and is more trusted.

This theme utilizes the power of a model and a tools framework to help maintain a common source of design data for handoff points between tools and between people. The model formalizes the information and the tools framework maintains the single source of data. Design artifacts such as software, documents, or tests are generated from the single source of data rather than re-entered or maintained by hand. Single source data error-proofs handoffs between tools and people and reduces the waste of over-processing that occurs when engineers re-enter design data.

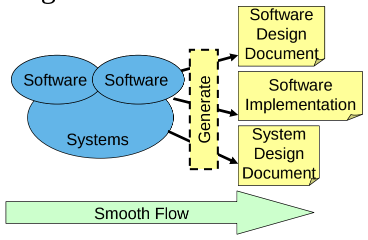

Fig. 10. Single Source of Data

Single source design data is easier to achieve when the number of handoffs between people, the number of handoffs between tools, and the overall number of tools is minimized. Each handoff is the waste of motion. "Keep it simple" rules the day. Teams should use the value stream to identify waste and to determine where tools can and should be applied.

4.6 "Good Enough Solution" or "Common Solution"

Traditional development teams select and customize tools and designs to meet the needs of their project. The tool and design customizations are tailored to meet the specific needs of the team and are worded in the vocabulary of the team. Other teams often re-develop the same solution along with their own unique customizations. Re-development is the waste of over-processing.

The "Common Solution" approach encourages engineers to find and support common tools and designs. The "Common solutions" is as much about what a development team should do as it is about what a development team should not do. Design and development teams should be the best at finding and re-using existing solutions in their products and in their MBe toolsets. Every tool or model that is re-developed by another team creates waste at the company level and is sunk cost to the project.

In an ideal world, the development of a design of similar applications is the assembly and tailoring of existing components and the development of new components. New components are published for reuse by other project team members and by other development project teams. In Fig. 11, the box on the bottom of the picture depicts a store of reusable components. The reusable components are found in a common reuse store and are available to assemble into the software and system design model.

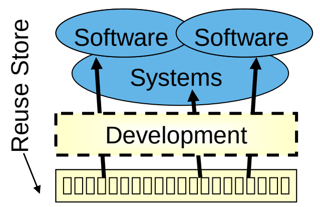

Fig. 11. Common Solution Reuse Store

Reuse of components requires design teams to embrace the concept of standard frameworks. Effective reuse requires both a standard design framework and a standard target framework. The standard design framework is the collaborative development environment enabling users to share design models back and forth and enabling models to be reused to develop new design models. The standard target framework enables models to be quickly generated and built for execution on the target system.

Each time a team implements design in the common frameworks they get better at understanding the issues and at predicting the schedule. Re-use opportunities can be capitalized as teams get better at developing and identifying the models. Common Solutions implemented in frameworks reduce unnecessary motion and over-processing caused when each development team solves the same problem in slightly different ways.

5 Case Studies

Early in the evolution of MBe at Rockwell Collins, MBe was used in a limited set of domains and was used to create very simple design artifacts. Feeling the pressure to reduce cycle time and costs, design teams began to leverage the characteristics of modeling to eliminate waste and improve design understanding. Executive leadership commitment to Lean principles enabled and accelerated this process. The following case studies describe the conception of MBe, the adoption of MBe by new teams, and the problems encountered in modeling. These real-world examples clarify the Lean principles enabled by MBe and describe how the MBe themes were identified.

The first case study describes how a development team evolved and adopted modeling techniques over the course of a long period of time. The second case study describes a project that started with no modeling experience or legacy and was able to quickly implement tools and methods for modeling. In both case studies the process was described and analyzed to remove waste. The resulting process and efficiency improvements are then described.

5.1 Case Study 1 – Flight Guidance Function

A Flight Guidance Function (FGF) uses a computer algorithm called a control law to command the flight path of an airplane. In simple terms, the FGF flies the airplane as an assistant to the pilot. The pilot sets guidance references that tell the FGF where to fly. The FGF flies the airplane to these references. For example, the pilot might direct the FGF to fly the airplane along a specific altitude (distance above the ground) and heading (direction).

The flight control function process has evolved over a number of years. The team members were using models when the initial process (a simplified value stream) was documented.

Because this is a case study, this section will describe the current value stream which is the process used to build systems before the MBe Lean initiative was started, and the future value stream, which is essentially the way the team builds systems today.

The current set of activities for the current value stream (the value stream at the start of the case study) is summarized as follows:

1. The systems analyst creates a design model and tests this model against functional and performance requirements.
2. The systems analyst then describes the design by writing it down as a design document and as detailed requirements.
3. The software engineer receives the design document and the requirements.
4. The software engineer updates the software design document and writes software.
5. The software engineer builds the system for testing on a test platform.
6. The software engineer performs functional and integration testing.
7. The systems analyst runs functional and performance testing.

These value stream activities are summarized in See Fig. 12.

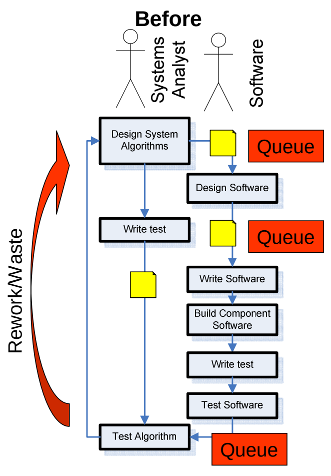

Fig. 12. FCF Current Value Stream Activities

Errors occur at the handoff between the systems engineer and the software engineer. Because the systems engineer runs tests after the software has been built, errors escape from systems engineering to engineering. Errors in the systems design cause the entire development cycle to be re-run. Queues form as errors accumulate, the design is changed and fixes are implemented.

Waste was identified in the form of defects, queues and over-processing. The future value stream was developed to reduce these three forms of waste. The future value stream is described as follows:

1. The systems analyst receives new requirements or change requests at the beginning of a design cycle.
2. The systems analyst develops work units for new functions or change requests.
3. The systems analyst develops test work units from the requirements and use cases.
4. The systems analyst creates the design in a model
5. The systems analyst tests the model and modifies the design until the model passes all tests.
6. The model is handed off between the systems analyst and the software engineer.
7. The software engineer creates work units for the new functions or change requests.
8. The software engineer creates or updates unit tests and hardware/software integration tests.
9. The software engineer updates the software integration model and the systems algorithm model.
10. The system analyst or software engineer generates the software directly from the models.
11. The system analyst or software engineer tests the software and modifies the design until the tests pass.
12. The software engineer integrates all changes and performs all tests at the end of the design cycle.

See Fig. 13.

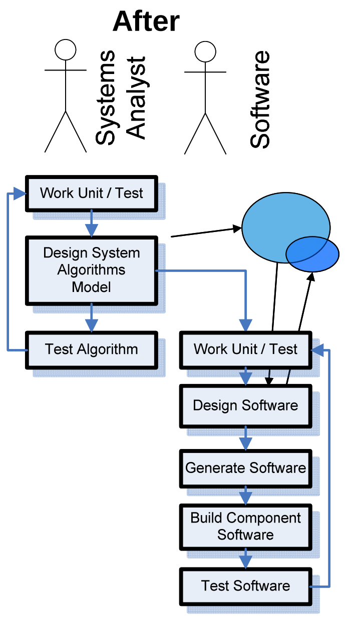

Fig. 13. FCF Future Value Stream Activities

The new value stream was implemented by the flight control team and did reduce the waste of defects, queues and over-processing.

The flight control team discovered the theme of small rapid design cycles called "Test-a-little Build-a-little". The original process took on the order of a day for a change to be designed, implemented, built, loaded into a test platform, and tested. The new process reduced this time to hours to test on the model, to build a system and to test the system on the target platform.

The flight control team also discovered the theme of single source design data. The software model was directly derived from the systems model. This reduced the effort involved in both translating an algorithm model first into a design document and then into software and also reduced effort in checking the consistency of the algorithm model and the software. The single source model resulted in fewer errors because the manual transfer of data from one format to another was replaced with a common model.

5.2 Case Study 2 – Flight Displays

A second example of using MBe to assist in Lean engineering is the flight displays group. Flight displays are integrated instruments that assist the pilot in flying an airplane. The flight displays team adoption of MBe was different than the flight control computer team's adoption for the following reasons:

1. The flight displays team knew very little about modeling technologies before adoption.
2. Flight displays tool chains were more complex because a graphical user interface model and the algorithm model tools needed to be integrated together.
3. The change was revolutionary as opposed to the flight control computer evolutionary change.

The value stream activities for the flight control function described the handoff of design data between different design teams. The displays group current value stream activities identified the same handoff between design teams. In addition to this they also identified a handoff between the Graphical User Interface (GUI) design and the Behavior design. This case study examines the handoff between the GUI design and the behavior design. The value stream activities describing the hand-off of changes between the GUI and Behavior design are described as follows:

1. A requirement is changed or added.
2. The software GUI engineer changes the software GUI portion of the design document.
3. The software behavior engineer changes the Behavior portion of the design document.
4. The software GUI engineer changes the GUI software.
5. The software behavior engineer changes the behavior software.
6. The tester changes the tests.
7. The tester builds the system for testing on a test rig.
8. The tester performs requirements based functional and integration testing.
9. A validator checks that the software is consistent with the design.

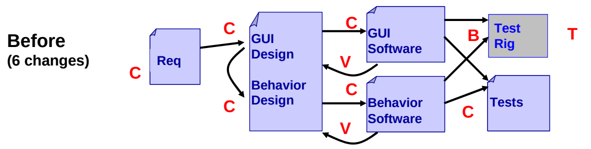

Fig. 14. Flight Displays Current Value Stream Activities

Errors occurred at the handoff between design, software, and test. Queues formed between GUI design and Behavior design, GUI design and GUI software, behavior design and behavior software, and between software and testing. Finally one change caused the update of requirements, GUI design, behavior design, GUI software, behavior software, and testing. The errors, queues, and numerous changes are examples of defect, inventory, waiting, and over-processing waste. Changes require hours to days to implement.

The future value stream eliminated or reduced these forms of waste by introducing MBe and a standardized interface definition. Interface requirements are captured in the standardized interface definition. The interface change can be updated or imported in the GUI and Behavior model. Tests are generated from the model and run on the simulator or on the target build. In the best case scenario an interface change that previously required the manual update of 6 artifacts, 2 validation steps, 1 build, and 1 test (See Fig. 14) is reduced to a manual update of 1 artifact, 3 automatic updates, 1 build, and 1 test. In the worst case, the 3 automatic updates require manual updates. See Fig. 15.

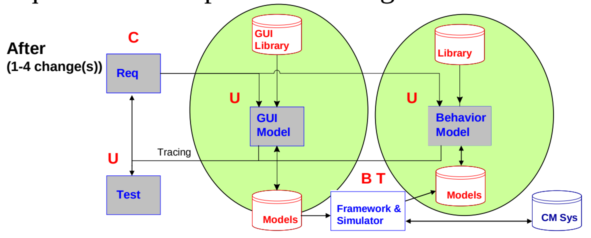

Fig. 15. Flight Displays Future Value Stream Activities

The future value stream activities are now as follows:

1. A requirement is changed or added to the formal interface definition.
2. The work unit is defined to implement changes or to add new features.
3. The engineer creates the manual test or generates automatic tests.
4. The software engineer updates the software GUI model (may require manual changes to the model).
5. The software engineer updates the Behavior model (may require manual changes to the model)
6. The tester builds the system for testing on a simulator or test rig.
7. The tester performs requirements based functional and integration testing.

The future value stream activities are streamlined and efficient. Simple changes to the interface can be accomplished by one person in minutes to hours.

6 MBe and Lean Benefits

In these two case studies, MBe and Lean waste reduction were applied to the processes and tools used by the teams. The teams identified the current process and the forms of waste. Using MBe technologies, the teams were able to reduce the waste of transportation, inventory, motion, waiting, over-production, over-processing, and defects. Focusing on the 6 themes of Lean: Customer Defined Value; Frontload Design; Pull-a-Little; Test-a-Little Build-a-Little; Single Source Design Data; and common solutions, new teams at Rockwell Collins are quickly learning and implementing the key strategies of Lean and MBe. Fig. 16 summarizes the trends teams see as they apply MBe and Lean.

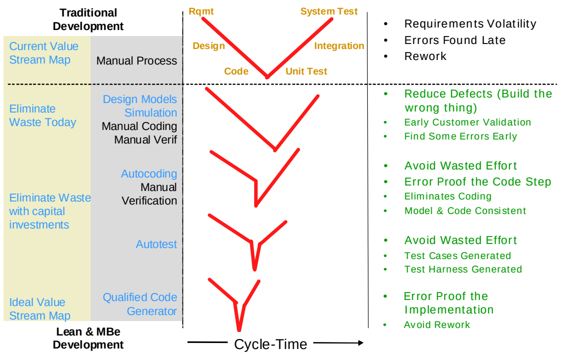

Fig. 16. Cycle Time Savings Trends with Lean and MBe

The result of this trend is both satisfied customers who receive products earlier and satisfied share-holders who receive increased stock prices because of cost reductions. How often does that happen?

Reference

[1] Womack J, and Jones D. *Lean Thinking*. 2nd edition, Simon & Schuster, 2003

[2] George M, Rowlands D, Price M, Maxey J. *The Lean Six Sigma Pocket Toolbook*, 1st edition, McGraw-Hill

[3] Morgan J, Liker J. *Toyota Production System.* 1st edition, Productivity Press, 2006.

[4] SYSML. *SysML – Open Source Specification*. http://www.sysml.org/

[5] Unified Modeling Language. *Object Management Group – UML*. http://www.uml.org/

[6] AADL. *The SAE AADL Standard Info Site.* http://aadl.info/

[7] MARTE. *The Official OMG MARTE Web Site*. http://www.omgmarte.org/

[8] IDEF Integrated Definition Methods. http://www.idef.com/idef0.html

[9] RTCA SC-167/EUROCAE WG-12. *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc. 1992

Copyright Statement