Effective Software Delivery Class – Capital One

Facilitator Guide

Dave Nicolette

Version 1.0.0 – July 21, 2015

Audience

Members of software development teams, infrastructure support teams, and other technical professionals, regardless of formal job title or traditional "role."

Duration

Two to four days

Facilitators are expected to trim and tailor content to fit the available time and meet the specific needs of each group of participants.

Room equipment

- One or two overhead projectors. Screens are preferable to projecting onto walls or whiteboards because of glare.
- A projection system with audio support is preferred if the facilitator intends to use videos with sound tracks.
- Moveable tables and chairs – depending on activities chosen by facilitators, it may be necessary to clear a large space in the middle of the room, organize participants into small groups to work at tables, or otherwise change the arrangement of the room.
- Flip chart pads with sticky paper and easels.
- Whiteboards.
- Dry erase markers.
- Wall space for posting sticky sheets and sticky notes.

Supplies

- Sticky notes

- Sharpies or similar markers
- Gherkin style exercise kit
- Tennis balls *
- Envelopes *
- Sheets of paper *
- Coins *
- Powerful Questions exercise kit: http://deborahpreuss.com/resources/DeborahPreuss_PowerfulQuestions_exercise_kit.pdf *

* depending on the exercises/activities the facilitators wish to use.

Learning Goals

Sufficient understanding of key ideas to enable the teams to move forward with the agile transformation under the guidance of coaches and mentors who will work directly with the teams after the training class.

Key learning areas:

- Capital One business goals and IT goals to be met by adopting changes in the way we work
- The meaning of Continuous Delivery in the context of Capital One IT systems, and implications for changes in organization, process, and technical practices
- The meaning, value, and limits of team self-organization
- Preparing ourselves to embrace change – Beginner's Mind and related concepts; “How we can” vs. “Why we can't” mindset, etc.
- Useful conceptual models – Agile, Lean, Systems Thinking, Queuing Theory, Theory of Constraints, Complexity Theory
- Concept of time – Throughput vs utilization; how time gets lost; impact of cross-team dependencies; impact of context-switching; focus on flow, etc.
- Overview of process frameworks in use – Scrum, Kanban, SAFe
- Release flow
- Managing cross-team and cross-ART dependencies

- Collaborative work spaces
- Distributed and dispersed teams
- Feature teams vs. component teams
- Full-stack developer and complex enterprise environments
- Effective communication
- Collaboration – pairing, swarming, mobbing
- Test-driven approach
- User Stories, epics, and different kinds of work items
- Personas
- Feature mapping / story mapping
- Earned business value
- Prioritization vs. scheduling
- ATDD – overview, work flow, tools
- Test automation pyramid – different kinds of automated checks, purpose, tools
- Gherkin – syntax, style, domain language
- Cucumber basics
- Ruby language introduction
- Regex basics
- Cucumber project setup
- Writing Cucumber steps
- Test-driving features at the acceptance level

Class Walkthrough

Facilitator may skip or compress certain sections based on available time and on the current level of knowledge of class participants.

Don't skip

The following sections are the core content and should not be skipped.

1. Introduction

4. Continuous delivery

6. Overcoming barriers to continuous delivery

7. Context-switching overhead

14. User Stories

17. General work flow

19. Definition of ready and done

20. Test automation period

21. How to get things done

22. Cucumber and ATDD

23. Gherkin language introduction

24. Gherkin style and domain language

25. Gherkin, Cucumber, and ATDD work flow

26. Ruby language introduction

27. Cucumber demo and ATDD walkthrough

28. Writing Cucumber steps

29. Test-driven development

31. Tie-back to participants' goals

32. Retrospective

1. Introduction

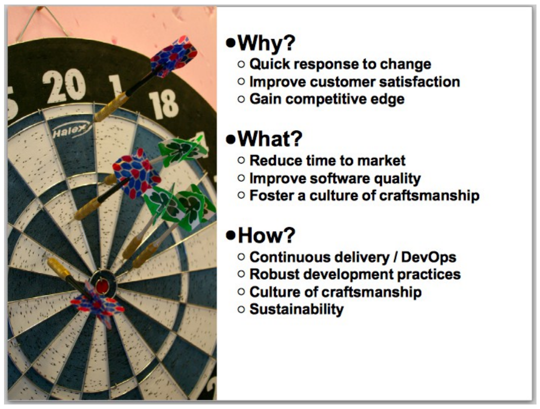
This section should not be skipped.

Goals:

1. Help participants understand the business drivers and organizational goals behind the transformation.
2. Help facilitators understand what participants want to learn in the class so they can adjust the content accordingly.

Facilitators may briefly introduce themselves and explain class logistics – break times, start/stop times, expectations regarding the use of electronic devices in class, location of rest rooms and break areas, etc. Facilitators may follow their own preferences in this regard.

Goal 1 – Understand business drivers and implications for IT



- **Why?**
  - **Quick response to change**
  - **Improve customer satisfaction**
  - **Gain competitive edge**
- **What?**
  - **Reduce time to market**
  - **Improve software quality**
  - **Foster a culture of craftsmanship**
- **How?**
  - **Continuous delivery / DevOps**
  - **Robust development practices**
  - **Culture of craftsmanship**
  - **Sustainability**

Business drivers

- Ability to respond quickly to customer feedback, changes in the market, competitive pressures, regulatory changes, and business innovation.
- Improved customer satisfaction to keep customers from switching banks and to encourage word-of-mouth advertising.
- Increase market share and overtake competitors that are currently ahead of Capital One, and maintain competitive advantage once it has been achieved.

What the IT department can do to support the business drivers

- Reduce time to market – shorten internal lead times from definition of intent to delivery of functionality. Can be achieved through a combination of organizational changes (team and ART structure and alignment with value streams), process changes (eliminating waste from the delivery pipeline), and technical practices (collaboration, automation, rigorous application of sound software design principles)
- Build in support for governance, legal, compliance, branding, UX, architectural, and performance concerns throughout the delivery process to eliminate after-the-fact reviews and approvals;
- Improve software quality – learn and use sound software design principles, share knowledge, implement automated functional checks at multiple levels, drive development from executable examples of system behavior; improve collaboration to ensure clarity of purpose and needs.
- Foster a culture of craftsmanship – reward actions that reflect a high standard of technical quality and professionalism; encourage low-ego, high-collaboration work practices; support the creation of internal technical communities of practice and events such as lunch-and-learns and code dojos.

Goal 2 – Understand what participants expect and/or want from the class

ACTIVITY

Provide pads of sticky notes and Sharpies (or similar) to all participants.

Facilitators

Place a sticky sheet on the wall and write the title, “Expectations,” at the top.

Participants

Working individually, write your expectations and learning goals for the class on sticky notes. Write one idea per sticky note. Write as many ideas as you want. Don't limit yourself to “ATDD.” Anything relevant to the transformation that you would like to discuss can be included.

Facilitators

Time-box the activity to 10 minutes. Ask participants to place their sticky notes on a sheet of sticky paper on the wall.

Arrange the sticky notes by affinity groups. Discuss each learning goal with the class so that everyone knows what expectations have been set by their peers. When a topic can't be included in the class, explain this and suggest an alternative way the participant can learn about the topic of interest.

Throughout the class as appropriate, connect your presentation with the expectations the participants listed during this activity. At the end of the class, revisit the expectations one last time and make sure each one has been addressed.

2. Self-organization

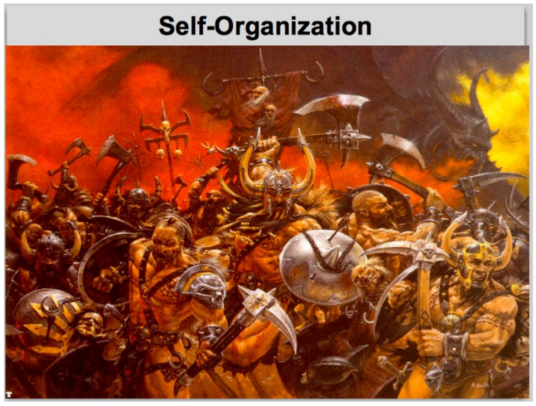
This section may be skipped at the discretion of the facilitators.

Relevance to Capital One

People tend to see “agile” and “Scrum” as a set of rules to be followed, rather than as a mindset. As a result, they tend not to make any decisions or attempt any changes unless they are *commanded* to do so by a manager. In that case, they focus on satisfying the demands of the manager to the letter, rather than thinking for themselves about how they can best achieve organizational goals. This habit must be broken if the transformation is to succeed.

Goals

- Understand the meaning of *self-organizing team*.
- Understand the limitations or boundaries of self-organization.
- Discover that self-organization is possible and not difficult.



Use an experiential learning approach to this topic. That is, participants first experience the effect in a controlled context in which they are slightly outside their comfort zone, and then facilitators debrief the experience to bring out the desired learnings.

ACTIVITY

Ball Point Game. See http://dpwhelan.com/blog/uncategorized/learning-scrum-through-the-ball-point-game/.

The point of the game is to demonstrate at a visceral level that teams can improve their delivery performance without much formality or planning.

Although the write-up credits Boris Gloger with inventing the game, in fact it was introduced (without any particular name) by psychologist Shamsuddin Butt at XP Days Germany in 2006. Boris later put a Scrum twist on it and named it Ball Point Game.

To play, the group should move tables and chairs as necessary to create a large open space somewhere in the training room. If the training room is too small to accommodate this, then move the group to another space – a lobby, conference room, or outdoors.

Debrief

Team self-organization in the context of agile software delivery means that a cross-functional team of professionals can determine how best to meet the business and technical goals of a sprint or release.

Self-organization does not mean the team makes business decisions about which intent to work on.

Self-organization is not the same thing as self-management. Technical teams are not expected to handle personnel issues and so forth without management support.

3. Conditioning the mind to embrace change

This section may be skipped at the discretion of the facilitators.

Facilitators are free to substitute their own activities or games to help participants achieve the learning goals.

Relevance to Capital One

People exhibit tremendous reluctance to attempt anything different than their current procedures and practices. This tendency remains strong even in the face of constant assurance by management that experimentation is desired and mistakes are acceptable provided they lead to learning. No matter how many training classes people attend, when they return to their everyday work context they discard everything they learned in class and simply continue to follow their familiar routine.

Goals

1. Get a gut feel for thinking differently
2. Understand that you are capable, personally, of thinking differently
3. Learn specific techniques to help yourself think differently

3.1. Beginner's Mind

Cultivate in yourself a beginner's mind
"In the beginner's mind there are many possibilities, in the expert's mind there are few."
- Shunryu Suzuki

As we gain experience in our careers, we become *experts* in one or more aspects of our work. One tendency of an expert is to apply the same solutions over and over again. This is because the expert has seen the same general pattern as the problem of the day, and applies the same solution that worked for them before. It is a side-effect of the human brain to apply a *first-fit* algorithm to find a solution, while we assume we are using a *best-fit* algorithm.

In contrast, a *beginner* does not have a mental catalog of previous solutions from which to draw. The beginner must reason through each problem and possibly experiment with multiple candidate solutions.

Shunryu Suzuki's concept of beginner's mind is to combine the best characteristics of the expert and the beginner. The *beginner's mind* does not forget all the information the expert has learned over the years, and at the same time is open to the beginner's unprejudiced approach to new solutions.

ACTIVITY

Experiential learning activity. Derive the underlying rule by testing various proposed solutions.

Facilitators

On a flip chart or whiteboard, write the following sequence of numbers followed by an underscore where the next number in the sequence may be written:

2 4 8 __

> **Important: Ask participants who have seen this exercise before to refrain from playing or giving away the answer. We want people to experience the *gut feel* of the exercise. The value will be lost if someone gives away the rule.**

The sequence of numbers follows a rule. We are not telling you what the rule is. You can suggest different numbers to fill into the blank. We will only say, “Yes, that follows the rule,” or “No, that doesn't follow the rule.”

Here's an example of the kind of exchange you might see:

> Participant: Does the number 16 work?
>
> Facilitator: Yes.
>
> Participant: OK, I think the rule is that the next number is calculated by multiplying the previous number by 2.
>
> Facilitator: That's not the rule. Try another number.
>
> Participant: Does the number 12 work?
>
> Facilitator: Yes.
>
> Participant: OK, I think the rule is that the next number is calculated by adding the previous two numbers together.
>
> Facilitator: That's not the rule. Try another number.

People can try as many numbers as they want before guessing at the rule. They can try again and guess again. After some reasonable time, end the exercise and debrief.

Debrief

The point of the exercise is *not* to guess the rule. The point is to experience the *feeling* of thinking like a beginner, and the *feeling* that results from assuming the solution to a problem is complicated when it really is simple.

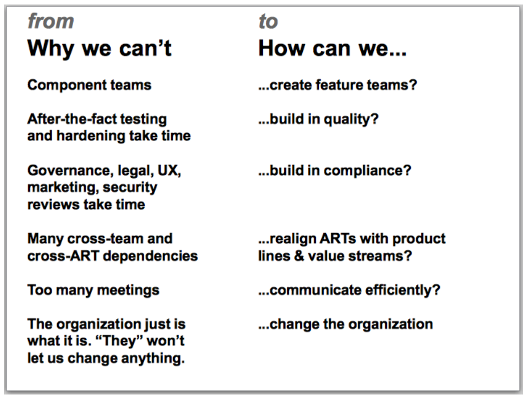
The rule is this: Each number must be larger than the previous number.

What normally happens is that adults recognize a pattern and they assume that pattern describes the rule. They suggest numbers that conform with their assumption and their numbers do in fact follow the rule, but not for the reasons the person assumes. When a number follows the rule but does not match the person's assumption, they usually try to work out a *more complicated* rule that will match all the numbers tested thus far. Most adults never think to check a case that deviates from their own assumption. They only seek to validate their assumption. This is *expert* thinking.

Most young children guess the rule very quickly. They have little or no past experience with sequences of numbers. They see that each number is larger than the previous one, and they make no assumptions that the underlying rule is complicated. This is *beginner* thinking.

The *beginner's mind* can apply the lessons learned from experience while remaining open to unexpected and often *simpler* solutions.

3.2. “How we can” thinking



from **Why we can’t**	*to* **How can we...**
Component teams	...create feature teams?
After-the-fact testing and hardening take time	...build in quality?
Governance, legal, UX, marketing, security reviews take time	...build in compliance?
Many cross-team and cross-ART dependencies	...realign ARTs with product lines & value streams?
Too many meetings	...communicate efficiently?
The organization just is what it is. “They” won’t let us change anything.	...change the organization

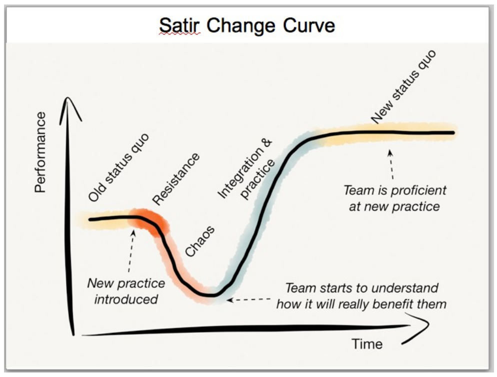
Most people at the company are quick to explain *why we can't* do things differently. To help the transformation succeed, we want to help people recognize when they are telling themselves *why we can't* and consciously change their thinking into *how we can*.

Facilitators

Lead a group discussion. Ask participants to state their reasons why a given practice is not practical in their work context. Use your judgment to determine whether their explanation is an example of *why we can't* thinking (it's possible they really can't apply the given practice, for a legitimate reason). When you get a good example, explore it with the group. Remind them to use their *beginner's mind.*

Typically, people assume that it is impossible to do things any differently than the way they are currently done. It is hard for them to think in terms of *how we can* rather than *why we can't*. In the discussion, guide people toward *how we can* thinking without suggesting concrete solutions to their problems. They need to learn to approach their own problems with a different mindset; they don't need to take specific suggestions from class instructors, only to turn around and explain *why we can't*.

3.3. Satir change curve



The curve is named for family therapist Virginia Satir, who identified the model in the context of working with families who needed to intervene in self-destructive behavior of a family member.

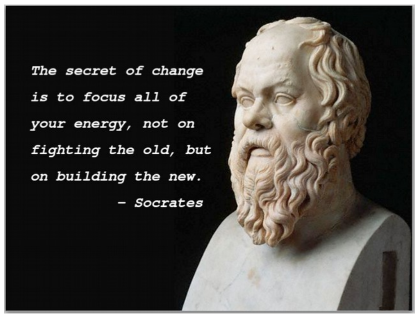
No matter what kind of change – a change in work practices, a change in lifestyle, overcoming an addiction – and no matter what kind of organization – a company, a sports team, a family – introducing change tends to follow the same pattern Dr Satir identified. When the *old status quo* is disrupted by a *new practice*, there is initial *resistance*. As the organization struggles with the new practice, there is a period of *chaos* when people aren't sure exactly what to do or what to expect.

During this period of *chaos,* performance is lower than the *old status quo*. The organization chooses whether to revert to the old way of working because it is comfortable and predictable, or to drive forward through the challenges to achieve a higher level of performance than before.

Having gained some experience with the new practice, the organization climbs the performance curve. Ultimately they reach a higher level of performance than before. As the new practice becomes ingrained, the organization settles into a *new status quo* at a higher level of performance.

The pattern can be repeated indefinitely to achieve *continuous improvement*.

The key challenge is to maintain discipline in using the new practice long enough to cross the trough of performance and start to climb out. To do this, people must stay focused on the *new* and leave the *old* behind.

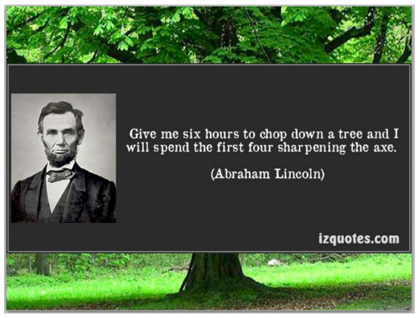

The secret of change is to focus all of your energy, not on fighting the old, but on building the new.
– Socrates

Sharpening the saw

Are you too busy to improve?
No thanks!
We are too busy

People often say they don't have time to work their way through the change curve because they can't slow down with delivery – not even temporarily, not even a little, not even occasionally.

The fatal flaw in this line of thinking is that delivery performance does not remain stable. Individuals, teams, and organizations that do not actively improve their practices lose their edge – their performance deteriorates. The truth is, you don't have time *not* to improve.


Give me six hours to chop down a tree and I will spend the first four sharpening the axe.
(Abraham Lincoln)
izquotes.com

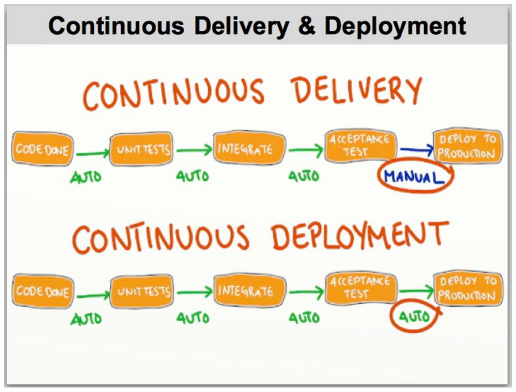
An analogy that is often used for this is the idea of sharpening the saw. If you continually cut wood with a saw, the blade gradually becomes dull. You must sharpen the saw from time to time even to maintain the *same* level of performance over time. You have *no chance* of improvement – or even of maintaining your current level of performance – if you never sharpen the saw.

4. Continuous Delivery

This section should not be skipped.

Relevance to Capital One

Continuous delivery is the primary technical goal of the IT organization. It supports the business drivers for the organizational transformation. Technical staff must understand the challenges and roadmap for achieving this goal.



Goals

1. Ensure participants understand what continuous delivery means in general and how it will help the company achieve the goals that were presented in the introduction.
2. Relate the concept of continuous delivery to Capital One's technical infrastructure and current state.
3. Help participants think about barriers to continuous delivery in their own work.

Overview

Depending on the current level of knowledge of the participants, you might want to provide a general overview of continuous delivery (CD). This is not a hands-on DevOps class, so be careful not to do a “deep dive” into any single facet of CD.

We want to aim for a flavor of CD that takes software changes all the way to production in a matter of minutes. This implies automated testing at multiple levels, automated promotion of code through multiple environments, automated monitoring of production environments to detect problems, as well as “baking in” governance, legal, and other considerations so that after-the-fact reviews and approvals can be eliminated. There are numerous technical and organizational challenges.

Identify barriers to continuous delivery

BARRIERS ACTIVITY – Part 1

Let participants use their beginner's mind to identify barriers to CD. Time-box this part of the activity to 10 minutes (don't count the time it takes them to get settled into their groups).

Facilitators

Hand out sticky notes and markers. Ask participants to self-organize into several small groups, with each group seated at their own table.

Participants

Think of barriers to CD in your current organizational and operational context. Write one barrier per sticky note. Discuss in your group how the barrier prevents software changes from flowing smoothly all the way to production without any stops. Don't try to solve the problems just yet.

Facilitators

Monitor the groups and remind them not to try and solve the problems they have identified. We want to get a long list of issues, not a deep dive into one or two issues.

At the end of the timebox, have participants place their sticky notes onto a sticky sheet on the wall. Organize the notes into affinity groups.

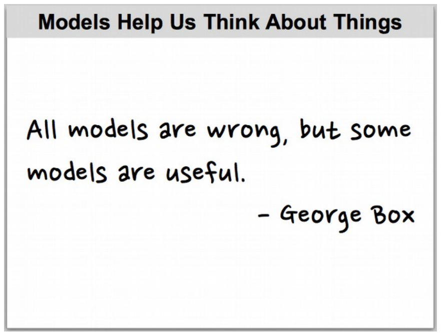
Invite participants to come up and read the barriers that everyone identified.

The second part of this activity will take place after certain key information has been presented.

5. Useful conceptual models for understanding software delivery

This section may be skipped at the discretion of the facilitators.

If participants are already well versed in these models, or if you are limited to a short time to present the class, then you may wish to skip or compress this section.



Models Help Us Think About Things

All models are wrong, but some models are useful.

– George Box

Box was writing about mathematical models when he published this statement in 1960, but the phrase has become commonplace when discussing models of all kinds.

The reason all models are “wrong” is that no single model completely describes reality. A model can only provide one perspective on reality. By learning to use many different models, we can achieve a more comprehensive understanding of reality than we can by focusing just on a single perspective.

The reason some models are “useful” is that they can help us understand one or more aspects of a complex situation in a way that enables us to take effective action to achieve goals.

It is good to learn to recognize when the usefulness of a model reaches its limit so that we can avoid becoming enamored of the model itself at the cost of missing our goals.

Relevance to Capital One

Most people in the company latch onto a specific branded framework or methodology and try to force everything they do to fit into that particular model. To succeed with the company's goals, we need to learn to examine the complexities of the business and the technical environment in multiple ways, and to be able to conceive of improvements and solutions that bring us closer to our *business goals* (responsiveness to the market, improved software quality, etc.) rather than closer to a *defined process* (Agile, Scrum, SAFe, etc.).

Goals

1. Understand how a conceptual model can highlight selected aspects of a complex system so that we can understand subsets of the system in a useful way.

2. Understand that conceptual models are limited in that no single model can possibly describe a complex system fully.
3. Identify a few conceptual models that are useful in the context of our effort to improve software delivery.

5.1. Agile Software Development

This model arose from work "in the trenches" of software delivery, and not as an academic exercise. It is based on observations by practitioners of things that "worked" vs. things that "didn't work." When something "worked" for them, they did more of it and tried to do it better. When something "didn't work" for them, they stopped doing it or did less of it.

From that simple and pragmatic beginning, a group of 17 respected software methodologists got together in February, 2001, and codified their observations of things that "worked" into a list of four value statements, known as the Agile Manifesto.

People often focus on the four value statements in isolation from the rest of the document. It is useful to remember the opening statement of the document:

We are uncovering better ways of developing
software by doing it and helping others do it.

Implications: "Agile software development" involves actively and mindfully thinking about *how we do our work* and continually *changing* the way we work to achieve better results. It is *absolutely not* about locking in a set of rules, ceremonies, and prescribed practices and then following them by rote.

Refer participants to: http://agilemanifesto.org

5.2. Systems Thinking

Systems Thinking is a way of reasoning about complex systems. Key tenets are:

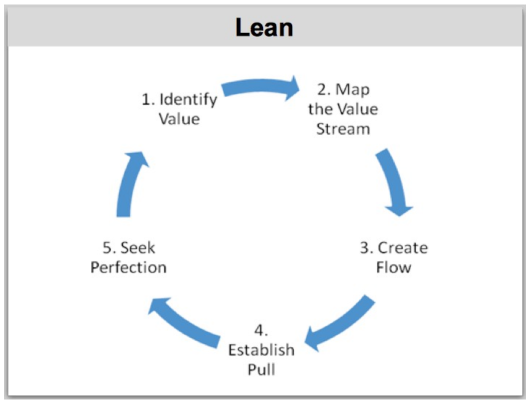
- The behavior of a system can't be understood by disassembling it into its constituent parts. The behavior of a system can only be understood by analyzing the dynamic interactions between its parts.
- The parts of a system are, themselves, systems. The universe comprises systems within systems within systems.
- Any organization of humans – such as a corporation – is a complex system.
- The possible behaviors of any component of a system is constrained by the systemic forces acting on the component.

- It is not possible to predict the outcome of changing any single variable in a complex system. The only practical way to effect improvement in, for example, an organization is to conduct a series of experiments and determine the effects of changes empirically. This approach is known as *sense and respond*.

Implications:

1. We must be willing to carry out experiments to effect improvement in our organization. Not all experiments will yield improvement, but all experiments will offer learning opportunities.
2. Barriers to achieving our goals are often systemic, and can't be solved by changing any single variable. We need to use *sense and respond* to find ways to move closer to our goals.

5.3. Lean Thinking



Lean Thinking is a way of understanding and optimizing any process that comprises multiple steps. It is based on methods created at Toyota in the 1980s which were later dubbed “Toyota Production System,” or TPS. Lean Thinking is not just a rebranding of TPS, but is derived in part from TPS.

This model offers useful ways of looking at how time is used in a software delivery process. It helps us identify places in our process in which the work stops while waiting for a dependency of some sort, whether it is a technical dependency, a review, or an approval. It also includes a narrow definition of “waste” that helps us focus on value-add activities rather than busy-work.

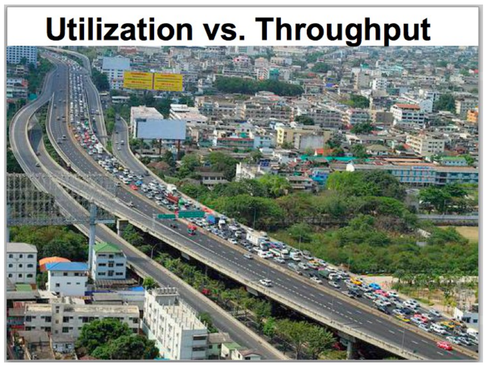
The basic values of Lean Thinking are:

1. Identify value – defined as anything a customer is willing to pay for
2. Map the value stream – understand the sequence of events that leads to the production of customer-defined value
3. Create flow – establish a process that enables work to move smoothly from start to finish without stopping

4. Establish pull – let customer demand drive the process by *pulling* results through it, rather than *pushing* output to customers whether they want it or not
5. Seek perfection – continually look for ways to improve the process; use the unattainable goal of "perfection" so that you will never feel complacent about your current state

5.3.1. Utilization vs. throughput

Lean Thinking offers a useful way of looking at the way time is used. A key concept is the difference between *resource utilization* and *throughput*. The concept is also part of another model, Theory of Constraints, described later.



When a resource operates at 100% of its capacity, it is said to be *fully utilized*. For example, the roadway on the right in the illustration has higher *utilization* than the roadway on the left. It has more tires per square meter than the roadway on the left.

The problem with this is the fact a roadway is not intended as a storage facility for tires. It is intended as a means for people to travel from point A to point B. Vehicles are not moving well on the roadway on the right. More vehicles are moving toward their destinations on the roadway on the left.

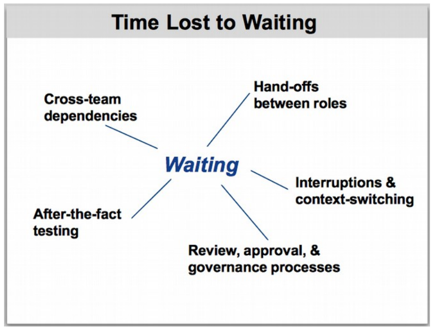
Throughput is defined as the number of *value units* delivered by a process per unit of time. If we think of a value unit in our software delivery process as a "feature," then to achieve the organization's goals for delivery performance we want to deliver as many features per release as possible. In other words, we want to maximize *throughput*.

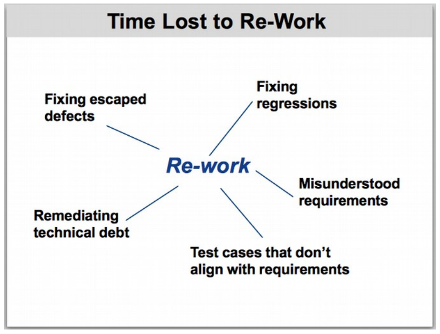
Utilization and throughput tend to be opposing forces. When we maximize utilization, we reduce throughput. To maximize throughput, we must be willing to allow some resources to operate at less than 100% of their capacity. This is counterintuitive to conventional management thinking.

Just as a roadway is not meant to be a storage facility for tires, a software delivery pipeline is not meant to be a storage facility for work in process. We want to *finish* features and get them into production, not merely to *start work* on many features in order to maximize our utilization. Lean Thinking is a useful model for us because it helps us see when we are blocking throughput by focusing on utilization.

5.3.2. Value-add vs. non-value-add time

Lean Thinking defines value-add time as any time spent advancing a product along the value stream toward delivery. Non-value-add time is the rest of the time work spends in the delivery pipeline before it is delivered.



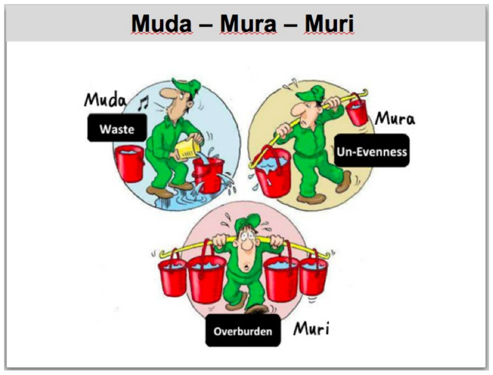

Time Lost to Re-Work
Re-work
Fixing escaped defects
Fixing regressions
Misunderstood requirements
Test cases that don't align with requirements
Remediating technical debt

ACTIVITY

Standing next to the sheet that lists the barriers to continuous delivery, read the items they identified and ask participants to relate them to the concept of value-add vs. non-value-add time. Ask participants if they can think of any additional barriers based on the concept of value-add time vs. non-value-add time. Write them down and add them to the sheet.

5.3.3. Muda, mura, muri

Lean Thinking borrows these three Japanese terms to describe characteristics of a process that can prevent high throughput.



Muda refers to any non-value-add work. For example, fixing software defects that should not have been created in the first place.

Mura refers to unevenness of flow. For example, stopping work in process in order to have it reviewed and approved, instead of building in the necessary elements in the first place.

Muri refers to overburdening a resource or a person. For example, asking a team to work excessive overtime to meet an arbitrary deadline, so that they will have to stop work afterwards to recover from exhaustion.

ACTIVITY

Standing next to the sheet that lists the barriers to continuous delivery, read the items they identified and ask participants to categorize them as muda, mura, or muri. Ask participants if they can think of any additional barriers based on this concept. Write them down and add them to the sheet.

5.3.4. Batch size and work-in-process

Lean Thinking considers *batch size* as a factor that affects throughput. Batch size is the number of work items that must be completed as a unit before product is delivered. *Work-in-process* is the number of work items that have been started but not yet completed. Let's see how these factors affect throughput and flow and then relate the information to our software delivery process.

ACTIVITY

Coin-flipping game

You can do this with participants in small groups, or with one group in the front of the room and the rest of the class observing. You need about 200 coins or some other kind of token that can be turned over. You need to keep track of various timings and you need a way to record the results (flip chart or white board).

Six to eight people form an assembly line. Dump the coins at one end of the line. Prepare the score card something like this:

	Round 1 - batch size 10	**Round 2 - batch size 1**
T0		
T1		
T2		
T3		

First round – batch size 10

1. First person flips 10 coins from heads to tails, then pushes the coins to the second person.
2. Second person flips the 10 coins from tails to heads, then pushes them to the third person.
3. Third person flips the 10 coins from heads to tails, then pushes them to the fourth person.
4. Etc.

The round is time-boxed at two minutes.

- T0 is the moment the first person flips the first coin.
- T1 is the moment the first batch of coins reaches the second person.
- T2 is the moment the first batch of coins reaches the last person.
- T3 is the moment the first batch is delivered at the end of the line.

Debrief – round 1

- How long did it take for the work to reach the second person in line?
- How long did it take for the work to reach the last person in line?
- How long did it take for a customer to see any results from the process?
- How many coins were left in the pipeline but not yet delivered?
- How did it feel to wait for the work to reach you during the exercise?

- Any other observations or comments?

Second round – batch size 1

5. First person flips 1 coin from heads to tails and pushes it immediately to the second person.
6. Second person flips 1 coin from tails to heads and pushes it immediately to the third person.
7. Third person flips 1 coin from heads to tails and pushes it immediately to the fourth person.
8. Etc.

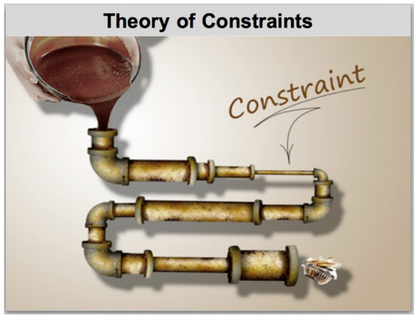
Debrief – round 2

- How long did it take for a customer to see any results from the process?
- How long did it take for the work to reach the last person in line?
- In which round did it feel as if the work was moving along more smoothly?
- How many coins were left in the pipeline but not yet delivered?
- How did it feel to wait for the work to reach you during the exercise?
- Any other observations or comments?

Standing next to the sheet that lists the barriers to continuous delivery, read the items they identified and ask participants to identify items they think are affected by batch size and/or work-in-process. Ask participants if they can think of any additional barriers based on this concept. Write them down and add them to the sheet.

5.4. Theory of Constraints

The Theory of Constraints (ToC), elaborated by Eliyahu Goldratt, posits that the capacity of any process comprising more than one step is limited to the capacity of the *constraint*; that is, the step that has the lowest capacity.



ACTIVITY – Part 1

Ask participants to self-organize into small groups at tables. Ask each group to write the names of several process steps on sticky notes, like this:

Requirements	Coding	Testing	Acceptance	Deployment

Ask participants to put numbers on the sticky notes to represent the capacity of each step, like this:

8 Requirements	10 Coding	4 Testing	9 Acceptance	6 Deployment

Distribute some of the coins or tokens you used in the coin-flipping game. Have participants move coins from one step to another in this process.

Ask participants to move coins through their process. Move the maximum number of coins that each step can process in each iteration.

For example, given the steps and capacities listed above, you would have this many coins in each step at the end of each of several iterations:

Iteration	Req'ments	Coding	Testing	Acceptance	Deployment
1	0	0	4	0	2
2	0	0	8	0	4
3	0	0	12	0	6
4	0	0	16	0	8

Debrief

1. Which step in this process is the constraint?
2. How many coins were delivered all the way through the process?
3. How many coins were left stuck in the process at the end of the last iteration?
4. What would really happen to the untested code that gets stuck in the Testing step? What impact might this have on the customer-defined value that is delivered?

ACTIVITY – Part 2

Repeat part 1 of the activity, but this time only allow participants to move 4 items at a time (the capacity of the constraint, Testing).

This plays out as follows:

Iteration	Req'ments	Coding	Testing	Acceptance	Deployment
1	4	0	0	0	0
2	8	0	0	0	0
3	12	0	0	0	0
4	16	0	0	0	0

Debrief

1. What was different in round 2?
2. How many coins were delivered all the way through the process?
3. How many coins were left stuck in the process at the end of the last iteration?
4. What is the next thing we could consider improving about this process?

ToC defines a mechanism to improve the delivery performance of a process known as the Five Focusing Steps. They are:

1. Identify the constraint
2. Exploit the constraint
3. Subordinate the rest of the process to the constraint
4. Elevate the constraint
5. Repeat

Identify the constraint means to find out the capacity of each step and locate the one that has the lowest capacity.

Exploit the constraint means to focus all available resources on helping the constraint perform at 100% of its capacity.

Subordinate to the constraint means to throttle back all steps upstream of the constraint to deliver output only at the constraint's capacity to process it.

Elevate the constraint means to improve the capacity of the constraint so that the throughput of the entire process can be increased.

ACTIVITY – Part 3

Replay the activity, but this time assume we have found a way to *elevate the constraint*. Set the capacity of the Testing step to 8 instead of 4.

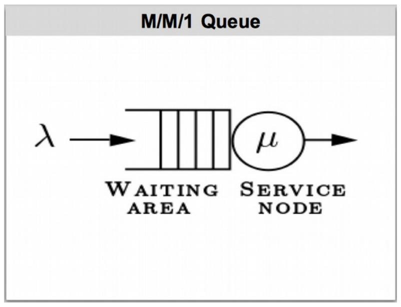
This plays out as follows:

Iteration	Req'ments	Coding	Testing	Acceptance	Deployment
1	0	0	0	0	2
2	0	0	0	0	4
3	0	0	0	0	6
4	0	0	0	0	8

Debrief

We used the Five Focusing Steps to elevate the constraint, Testing. When we do step 5, *repeat*, we discover a new constraint. Which step is now the constraint?

5.5. Network queuing theory



Queueing theory is a mathematics-based discipline in computer science that helps us understand how best to configure and manage networks. It is also useful for understanding non-computer queuing problems, such as processing customer service requests.

A queue is described in terms of its arrival rate (the rate at which new items arrive), service time (the average time it takes to process a work item), and service nodes (the number of processing nodes that can service work items).

It is a useful model for understanding software delivery processes, because such a process can be seen as a queuing problem. Software delivery processes typically have a variable *arrival rate*, a variable *service time*, and one *processing node* – namely, the delivery organization).

A queue that has those characteristics is called an M/M/1 queue, as a kind of shorthand that network folks understand. The math behind an M/M/1 queue shows us that maximum throughput is achieved when the queue is loaded at 70% of its capacity.

Discussion questions for the class:

Does your team commit to work based on 100% of its capacity? If so, do you think this slows things down or speeds things up? Why?

6. Overcoming barriers to continuous delivery

This section should not be skipped.

Autonomous, self-directed work in this area at the grassroots level is a critical success factor for achieving the organization's primary transformation goal.

Relevance to Capital One

This is directly related to making progress toward continuous delivery.

Goals

1. Encourage participants to apply their beginner's minds to the barriers they identified to continuous delivery
2. Share ideas about overcoming barriers to continuous delivery
3. Help participants identify something they can change immediately to help “move the needle”

BARRIERS ACTIVITY – Part 2

Now it is time to revisit the barriers to continuous delivery that participants identified earlier in the class. Armed with the information they have learned about various models, they can brainstorm potential solutions or experiments to overcome these barriers.

Time-box this part of the activity to 15 minutes.

Facilitators

Let participants self-organize into groups of people who share an interest in discussing one or two of the problems that were identified in part 1 of the activity. People need not stay in the same group they were in for part 1.

Participants

Brainstorm possible solutions or experiments that may solve or mitigate the problems your group has chosen to focus on. Remember to use your *beginner's mind* and remain open to unorthodox or radical ideas.

Facilitators

Monitor the groups and be alert for people slipping into *why we can't* thinking.

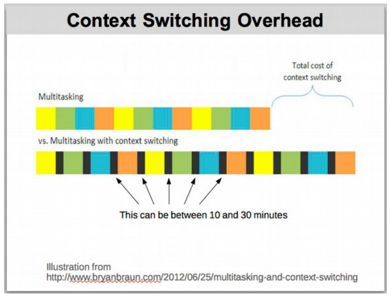
At the end of the timebox, ask a representative from each group to tell the class what ideas their group came up with to address the challenges they chose to discuss.

When everyone has had a chance to explain their ideas, ask participants to think about *one thing* they can change when they get back to their everyday work environment that will make progress toward eliminating one of the barriers to CD that they identified in the activity. Be alert to *why we can't* thinking.

7. Context-switching overhead

This section should not be skipped.

A significant portion of lost time is due to poor time management at the individual and team level. It is not recommended to skip or compress this topic.


Context Switching Overhead
Total cost of context switching
Multitasking
vs. Multitasking with context switching
This can be between 10 and 30 minutes
Illustration from http://www.bryanbraun.com/2012/06/25/multitasking-and-context-switching

Relevance to Capital One

Almost universally, technical professionals at this company are interrupted continually throughout the day. Common reasons include:

- Trying to juggle more than one task at a time (multitasking)
- Being pulled into meetings at arbitrary times (manager's schedule vs. maker's schedule)
- Being instantly available via IM or email at all times (poor time management)

People need to learn how to control these things and manage their time properly.

Goals

- Impact of multitasking and context-switching overhead
- Concept of manager time vs. maker time; correct concept of “core hours”
- Personal and team-level time management techniques

Choose one of the following activities or substitute one of your own that teaches the same lesson.

ACTIVITY – Stuffing envelopes

Facilitators

Give each participant ten sheets of letter size printer paper and ten standard number 10 envelopes. Participants will fold each sheet of paper in thirds and stuff it into an envelope.

Keep track of the total time it takes for everyone to stuff all their envelopes.

First round

Participants must follow this procedure exactly. To be sure it's clear, demonstrate it for them.

1. Make the first fold in each sheet of paper. No stacking! Fold each sheet separately. *Only* the first fold.
2. Start again with the first sheet of paper and make the second fold *only*, one sheet at a time.
3. Start again with the first sheet of paper and stuff it into an envelope. Stuff the envelopes one by one.
4. When you have finished, raise your hand and keep it raised.

Note the total elapsed time from the start of the exercise until the last person raises their hand.

Second round

Give each participant ten more sheets of paper. Have them remove the folded sheets from their envelopes.

Participants must follow this procedure exactly. To be sure it's clear, demonstrate it for them.

1. Take the first sheet of paper, fold it into thirds, and stuff it into an envelope.
2. Repeat with the second sheet of paper and envelope.
3. Continue in the same way until you have stuffed all ten envelopes.
4. When you have finished, raise your hand and keep it raised.

Note the total elapsed time from the start of the exercise until the last person raises their hand.

Debrief

Compare the total elapsed times from rounds one and two. As participants how it felt to complete the task using each procedure.

What normally happens is that it takes longer to complete the task in the first round than the second. This is because of time lost to context switching in the first round. In the second round, people focus on completing one thing at a time.

7.1. Context switching overhead

ACTIVITY – Columns of numbers

Facilitators

Give each participant two sheets of letter size printer paper and make sure everyone has something to write with. Be careful when explaining the procedure, because people tend to race ahead and do what they *think* they will be asked to do. If anyone does this, give them a fresh sheet of paper and remind them not to do anything until the timer starts. You might want to demonstrate the procedure for each round on a whiteboard or flipchart so they understand what to do.

First round

Write three columns of numbers and letters. Column 1 will be Arabic numerals from 1 through 20 inclusive. Column 2 will be Roman numerals from 1 to 20 inclusive. Column 3 will be the first 20 letters of the English alphabet, capitals, in alphabetical order starting with A.

Participants must follow this procedure exactly:

1. Write the first number in column A, then the first number in column B, then the first letter in column C.
2. Write the second number in column A, then the second number in column B, then the second letter in column C.
3. Write the third number in column A, then the third number in column B, then the third letter in column C.
4. ...and so on until the time-box expires.

Time-box the activity to one minute.

Second round

Using the second sheet of paper (don't continue with the first sheet), write three columns of numbers and letter in the same pattern as in the first round.

Participants must follow this procedure exactly:

1. Write all 20 numbers in column A.
2. Write all 20 numbers in column B.
3. Write all 20 letters in column C.

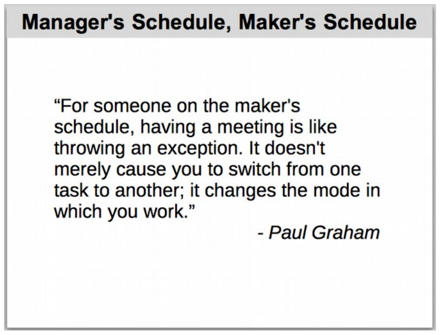
Time-box the activity to one minute.

Debrief

Compare the amount of work completed in each round.

What normally happens is that people complete more in the second round than in the first. The reason is the context-switching overhead in the first round when people have to remember which number or letter to write next. In the second round, people automatically write the numbers or letters in the sequence they know so well, without having to think about it.

7.2. Manager's schedule, maker's schedule, and “core hours”



Manager's Schedule, Maker's Schedule

“For someone on the maker's schedule, having a meeting is like throwing an exception. It doesn't merely cause you to switch from one task to another; it changes the mode in which you work.”
- Paul Graham

Paul Graham came up with the idea of the manager's schedule and maker's schedule. The idea is that managers divide the work day into one-hour time slots that they use for meetings. Many managers don't understand the impact to interrupting creative work at such frequent intervals.

Makers – that is, people who make things (like software) – work on a very different schedule. They require large blocks of uninterrupted time to focus on creative work. When this work is interrupted once in the middle of the morning for a meeting, and once again in the middle of the afternoon for a meeting, the entire day is all but lost.

It's important for technical team members to make themselves available for meetings at times when they are not engrossed in focused creative work. This is a question of individual and team-level time management.

One popular way of handling this on agile teams is to define one or two segments of the work day as *core hours*. During core hours, everyone on the team is focused on the work items the team is currently working on.

Unfortunately, many people at this company have a very misguided idea of what “core hours” means. They think it means one or two hours, one or two times per week, when the team has a “meeting” to “discuss” work in progress. This is completely wrong.

Core hours are the *maker's schedule* for teams to collaborate on completing their work. This does not imply “coding only” or “no meetings at all.” It includes any and all work that is relevant to completing the stories currently in play.

Email and IM are not open during core hours, except as needed to complete the stories currently in play. When a team member receives a request via email or IM from another team, they can answer it after core hours.

8. Process review

This section may be skipped at the discretion of the facilitators.

Relevance to Capital One

Many teams at Capital One do not function in accordance with the process framework they are ostensibly using. In some cases, basic understanding of the process seems to be completely absent.

If you are teaching a group that fits this description, consider providing a brief review of how planned and unplanned work is supposed to be handled using Scrum and/or Kanban.

You might hear participants say they understand the "theory" of the process, but they can't follow it for one reason or another (often because of fixed-date intent). This response reflects two things that need to be corrected:

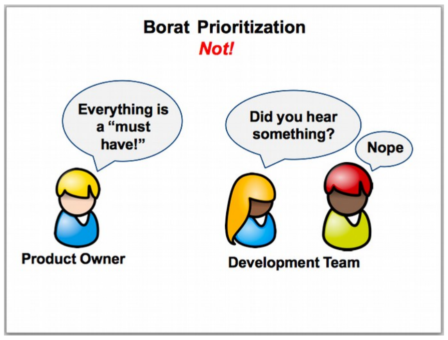
1. *Why we can't* thinking, which people should have learned about earlier in the class; and
2. A poor understanding of "agile" development, which is specifically designed to help organizations hit fixed delivery dates.

You may have to adapt your presentation to tailor the information to the current level of understanding of the group you're teaching.

Goal

1. Reinforce basic understanding of how Scrum and Kanban manage planned and unplanned work

There are several slides with content for this topic, starting with this one:



9. Transparency

This section may be skipped at the discretion of the facilitators.

Relevance to Capital One

Transparency is one of the most fundamental principles in the approach we are aiming for at Capital One. Yet, most teams never inform their Product Owner or other stakeholders when they are over capacity or when a story is at risk of being left incomplete at the end of a sprint.

It's very important to be open about these issues so that the appropriate business decisions can be made early enough to make a difference.

Goal

1. Help participants understand the importance of transparency

Play the following video from Youtube:

https://www.youtube.com/watch?v=NkQ58I53mjk

This is a scene from an old Lucy TV show. In the scene, Lucy and Ethel are working at a candy factory. Their task is to wrap each piece of candy that comes off the production line before it goes into the packing room.

At first, they can handle the candy moving past them on the conveyor belt. The supervisor sees that they are doing well, and increases the speed of the belt. Now the pair can't handle the amount of work they are receiving. Instead of being *transparent* about that, they hide the candy that they have been unable to wrap.

When the supervisor checks again, she sees that apparently the women are able to handle the workload. She increases the speed again.

Debrief

When you tell stakeholders that you are able to handle the amount of work they ask for, they believe you. They have no reason to think you can't handle it, unless you fail to deliver two or three times in a row, and then it's bad news for you.

Earlier you learned that your software delivery process is similar to an M/M/1 queue. That means 70% capacity is the optimal load factor for your team. When you are asked to complete 71% of your capacity, you are overloaded. You need to tell people immediately.

When you are tracking progress during a sprint and it looks as if you will not finish all the stories in plan, you need to tell people immediately so they have as much time as possible to deal with whatever issue the delay will cause. If they don't find out until the last minute, it's too late for them to do anything about it.

10. Cross-team and cross-ART dependencies

This section may be skipped at the discretion of the facilitators.

Relevance to Capital One

As in many large IT organizations, it isn't always feasible to put together a “feature team” that has all the skills and resources necessary to deliver an end-to-end feature. Adding to the very real logistical considerations, some people don't seem to understand just what a “feature team” is, or even what a “feature” is. Many teams call individual tasks “features” so that they can qualify for the label, “feature team.”

Whether the reason is logistical or conceptual, there are times when we just gotta cross the streams.

Goals

1. Practice working collaboratively to brainstorm solutions
2. Practice applying lessons learned about conceptual models, time, and waste
3. Practice taking responsibility for their own results, rather than just following orders and rules
4. Think of at least one action they can take to help manage cross-team or cross-ART dependencies in their own work

At this point in the class, participants have at least a basic idea of key concepts like Beginner's Mind, Agile, Lean Thinking, and Systems Thinking. In this activity, we let them apply what they have learned to the problem of managing cross-team and cross-ART dependencies.

If necessary, remind them that they have the tools they need:

- Beginner's Mind lets them think of unconventional and out-of-the-box ideas.
- Agile thinking means they are responsible for finding ways to streamline their work, rather than waiting for “someone” to tell them how to manage dependencies.

- Lean Thinking arms them with an understanding of throughput vs. utilization and various ways in which time can be wasted.
- Systems Thinking helps them understand that there won't be a single, simple change that magically fixes everything; they will have to *sense and respond* to systemic forces.
- They have enough knowledge to think of at least *some* actions they can take to mitigate the impact of dependencies.

ACTIVITY

Time-box the activity to 20 minutes, with 10 minutes for discussion afterwards.

Participants self-organize into small groups and work together at tables. They brainstorm ideas to help their teams mitigate the waste that is introduced when they have cross-team and cross-ART dependencies.

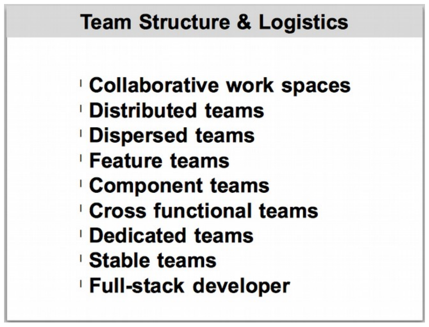
If possible, they think of at least one concrete action they can take when they return to their everyday work environment to mitigate this type of waste.

Debrief

Let a representative from each table summarize their results for the rest of the class. The class can discuss any of the ideas presented.

11. Team structure and logistics

This section may be skipped at the discretion of the facilitators.



Team Structure & Logistics

- Collaborative work spaces
- Distributed teams
- Dispersed teams
- Feature teams
- Component teams
- Cross functional teams
- Dedicated teams
- Stable teams
- Full-stack developer

Relevance to Capital One

The company has a variety of different team structures. It's helpful to understand how best to organize teams for best results. It's not always possible to do so, however, so it's also helpful to understand what the impact may be of different sorts of team structure.

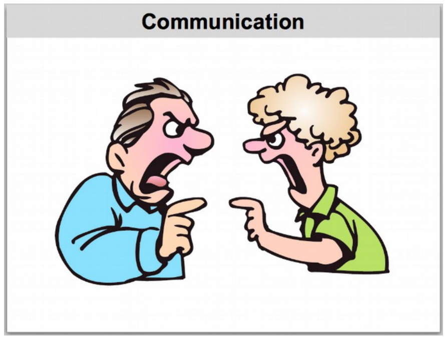
Goals

1. Understand how the physical work space affects collaboration
2. Understand the impact of distributed teams and dispersed teams on agility
3. Understand what a feature team actually is and how they can be formed at Capital One
4. Understand when a component team is necessary and how it can affect agility
5. Understand what a dedicated team is and how it affects agility
6. Understand when a stable team is desirable and when virtual teams are usefulness
7. Understand what "full stack developer" means and how the concept applies in a large-scale corporate IT environment

This is mainly a presentation and discussion topic. There are no activities.

12. Communication

This section may be skipped at the discretion of the facilitators.



Relevance to Capital One

Meeting the business goals of the IT transformation requires a collaborative working style. Effective collaboration requires good communication skills. Technical professionals don't usually receive any training in “soft skills.”

Goal

1. Get basic exposure to several effective communication techniques

12.1. Active Listening

Active Listening is a specific communication technique. In these two videos of scenes from the TV series, *Everybody Loves Raymond*, Raymond learnes the technique in an effective parenting workshop and uses it well.

1. https://www.youtube.com/watch?v=aP55nA8fQ9I
2. https://www.youtube.com/watch?v=4VOubVB4CTU

12.2. Powerful Questions

Some kinds of questions tend to shut people down while others encourage people to open up. Powerful Questions is a communication technique that you can use to improve the effectiveness of the questions you ask.

Use Deborah Hartmann-Preuss' Powerful Questions exercise described here: http://deborahpreuss.com/resources/DeborahPreuss_PowerfulQuestions_exercise_kit.pdf

13. Collaboration

This section may be skipped at the discretion of the facilitators.

Relevance to Capital One

Everyone at the company will agree that collaboration is a necessary and fundamental part of the new approach to software delivery. Yet, most of the technical staff has a poor understanding of what collaboration actually looks like in day to day, hour by hour, minute by minute work.

Goals

1. Understand when it is advisable to pair and when it is advisable to work solo
2. Learn about different styles of pairing and when each one may be helpful
3. Learn about a style of group collaboration called "swarming"
4. Learn about a rigorous form of whole-team collaboration called "mob programming"

13.1. Pairing

Pairing originated as *pair programming*, in which two programmers work together on the same development task at the same time using a single workstation. Pair programming was so helpful in software development organizations that the idea of working in pairs was extended to other development disciplines besides programming.

It's often helpful for team members to pair when working on any task that contributes to the sprint goal. An analyst might pair with a tester; a tester with a programmer; a programmer with a member of a team on which his/her team has a dependency. Any combination may be beneficial, depending on context.

There are nuances to pairing that might take a full day to cover adequately, using role playing and other learning techniques. That level of coverage is out of scope for this class. It's sufficient if you explain to the participants when and why to use various pairing styles, such as:

- Driver / Navigator

- Ping Pong
- Silent Running
- Evil Pair
- Silent Evil Pair

Briefly review the situations when pairing yields benefit and when it has potential pitfalls, such as:

- Senior person and junior person – knowledge transfer
- Two technical peers – reduced defects, improved designed
- One person thinks they are more senior than they really are – pairing will be less effective than solo work
- ...and other situations from your own experience.

13.2. Swarming

“Swarming” means that the whole team, or a subset of the team, focuses on the same story at the same time. Every individual in the group is not necessarily working on the same task, and everyone may not be pairing, but the whole team is in the same room where they can communicate easily and everyone is contributing to the same story.

13.3. Mob Programming

Mob Programming is a specific whole-team collaborative development method developed by Woody Zuill. It has been gaining popularity in the industry in the past few years.

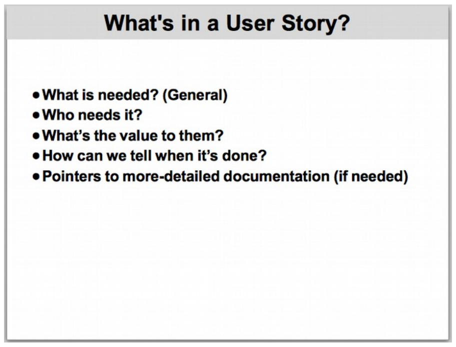
To give participants a sense of how it works, you can play the video at

http://mobprogramming.org/mob-programming-time-lapse-video-a-day-of-mob-programming/

This is a three-minute time-lapse video of a day in the life of an actual development team that “mobs” all day, every day.

14. User Stories

This section should not be skipped.



What's in a User Story?

- What is needed? (General)
- Who needs it?
- What's the value to them?
- How can we tell when it's done?
- Pointers to more-detailed documentation (if needed)

Relevance to Capital One

Most teams have no idea what a User Story actually is. They call any type of work item a "User Story" regardless of scope or context.

There is a common misconception that information radiators such as card walls and Kanban boards serve exactly the same purpose as electronic tools like VersionOne. This misconception is crippling the organizational improvement effort.

Goals

- Understand what a User Story is
- Understand what information a User Story should contain
- Understand that there is no rigidly-defined format for a User Story
- Understand that different types of work can be defined appropriately; there is no value is trying to force-fit everything into the form of a User Story
- Understand the difference between an information radiator and an electronic tracking tool

What's a story?

Give the general definition of a User Story – an interaction between a user and a system that provides some sort of value to the user.

Mention that in Capital One's current state, there are no true feature teams. Therefore, no team can work on a true User Story. Instead, the User Story concept applies at the "epic" or "feature" level. Individual teams will be working on items subordinate to an epic or feature. These subordinate items probably will not lend themselves naturally to the User Story format. That is okay.

Describe the kinds of information a User Story should contain – what, who, value, definition of done, pointers to details.

Describe the kinds of information a User Story should not contain – standard, common information that applies to multiple stories; details such as lists of codes and so forth.

Mention that the “stories” a team works on need not be true User Stories. As long as the work items are described in a way that helps the team complete useful work, any format is fine.

Story cards and information radiators

Agile teams are supposed to work in collaborative work spaces. Most of the practices typically associated with the word “agile” assume this is the case. When teams don't work in appropriate collaborative work spaces, some common “agile” practices might not make sense.

A collaborative team space has wall space where *information radiators* can be set up. An information radiator provides up-to-date status information at a glance to anyone who happens to look at it. Because it *radiates* information, there is no need to “sign in” and perform searches to find the information you need. You just look up, and there's the information.

In contrast, an electronic tracking tool is suitable for storing an arbitrary amount of detailed information about work in process and planned work. It contains substantially more information than an information radiator, but it is in the nature of reference information rather than tactical information of immediate use to the team minute-by-minute throughout the day.

Your team's card wall will display a card or sticky note for each story in process. These cards are not meant to include every detail about the story. They are meant to be placeholders.

They show the current state of the story, because they appear on the wall under a particular column, such as “In Progress” or “Ready for Acceptance.”

They show any blocks on a story, because color-coded stickers are applied to the cards when there is a block. That way, all team members and all other interested parties can see at a glance that a story is blocked, as opposed to having to log into VersionOne and explicitly search for stories to see whether they are blocked.

They serve as placeholders for conversations about the story. This encourages direct collaboration and discourages isolated, solo work based on unreliable written “specs.”

Story card wording

The exact wording on a story card is not important, provided the card is understood by team members.

Some popular formulae are:

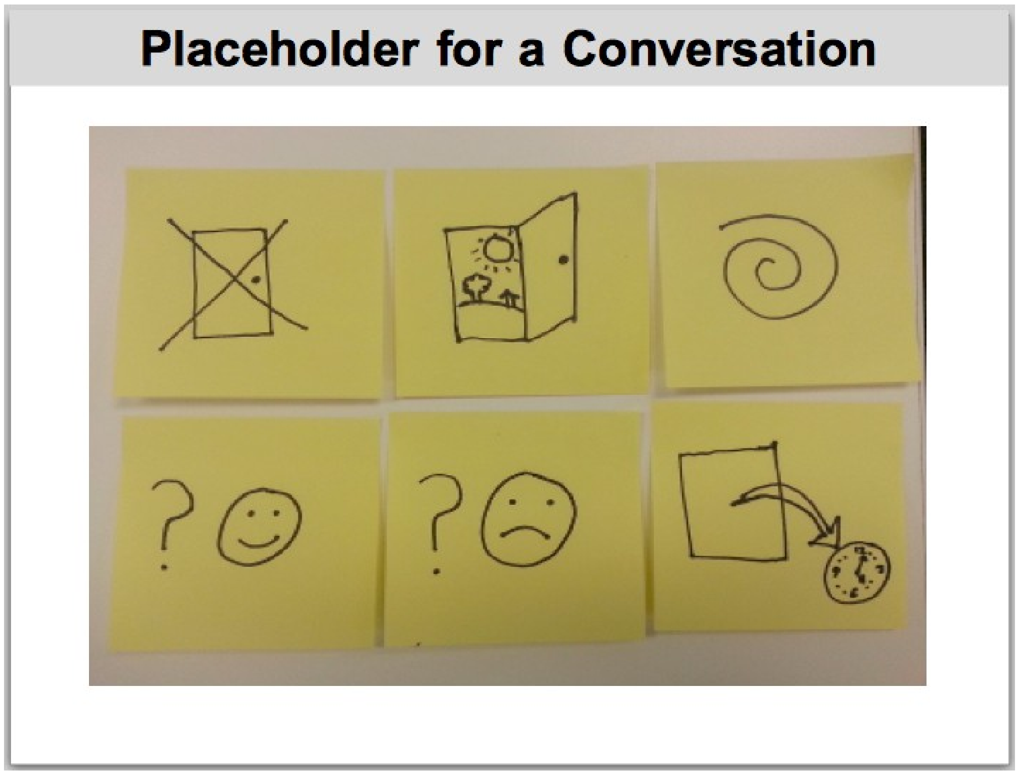
- As a <role>, I want <thing> so that <value>
- In order to <goal>, as a <role/system> I want <thing>

These templates are not “rules” that must be followed. A story card can just contain a one-line description of the work in any format at all.

A story card doesn't need any words at all, provided it serves its purpose as a placeholder and conversation-starter that all team members understand in a consistent way.

ACTIVITY

Ask participants to guess what sort of software functionality these story cards might represent.



It makes no difference if people guess “right.” The point is just that the story cards can contain whatever shorthand information is meaningful to that particular team.

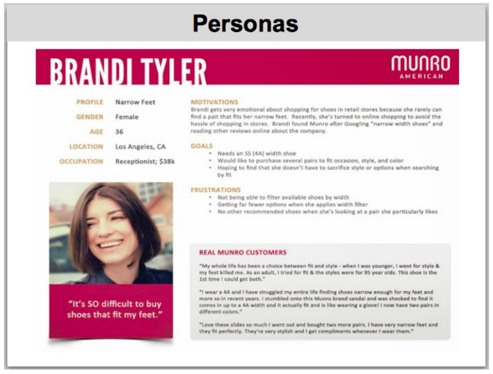
In case you're curious, the cards are meant to represent the following stories:

- Unsuccessful login

- Successful login
- Undo / redo
- Application approved
- Application denied
- Timeout handling

15. Personas

This section may be skipped at the discretion of the facilitators.



Personas

BRANDI TYLER

MUNRO AMERICAN

PROFILE	Narrow Feet
GENDER	Female
AGE	36
LOCATION	Los Angeles, CA
OCCUPATION	Receptionist; $38k

MOTIVATIONS

Brandi gets very emotional about shopping for shoes in retail stores because she rarely can find a pait that fits her narrow feet. Recently, she's turned to online shopping to avoid the hassle of shopping in stores. Brandi found Munro after Googling "narrow width shoes" and reading other reviews online about the company.

GOALS

- Needs an SS (4A) width shoe
- Would like to purchase several pairs to fit occasion, style, and color
- Hoping to find that she doesn't have to sacrifice style or options when searching by fit

FRUSTRATIONS

- Not being able to filter available shoes by width
- Getting far fewer options when she applies width filter
- No other recommended shoes when she's looking at a pair she particularly likes

"It's SO difficult to buy shoes that fit my feet."

REAL MUNRO CUSTOMERS

"My whole life has been a choice between fit and style - when I was younger, I went for style & my feet killed me. As an adult, I tried for fit & the styles were for 95 year olds. This shoe is the 1st time I could get both."

"I wear a 4A and I have struggled my entire life finding shoes narrow enough for my feet and more so in recent years. I stumbled onto this Munro brand sandal and was shocked to find it comes in up to a 4A width and it actually fit and is like wearing a glove! I now have two pairs in different colors."

"Love these slides so much I went out and bought two more pairs. I have very narrow feet and they fit perfectly. They're very stylish and I get compliments whenever I wear them."

Relevance to Capital One

Many teams have difficulty identifying how the work they do affects Capital One external customers or their own internal stakeholders. In part this is due to the fact that teams are organized around components or platforms rather than aligned with value streams, and the "intent" comes to them as a context-free stream of *ad hoc* technical tasks.

As we move toward feature teams and collaborative, cross-functional work, it will become more important for all contributors to a feature to understand who is affected and how the solution design might provide a better or worse user experience. *Personas* are a way to represent idealized real users.

Personas are also useful when automating acceptance tests, as the examples can be expressed from the point of view of particular personas, making them more realistic and representative of customers. When implementing the steps for Cucumber scenarios, it is easy to map persona definitions to reusable helper methods that simplify development and ensure consistent behavior throughout the automated test suite.

Goals

1. Understand the difference between a user type (role) and a persona
2. Understand what information is included in a persona
3. Understand how personas relate to discussions of intent
4. Understand how personas contribute to user experience design

ACTIVITY

Ask participants to self-organize into small groups to work at tables. Provide them with sticky notes and markers.

Given a hypothetical application to allow customers to apply for credit lines, write down the user types or “roles” that might use the application on sticky notes, one per sticky note.

Discuss in your group different people who fall into each “role” category who have different lifestyles, accessibility needs, or personal preferences. Give each one a name. Line them up under the “role” where they belong.

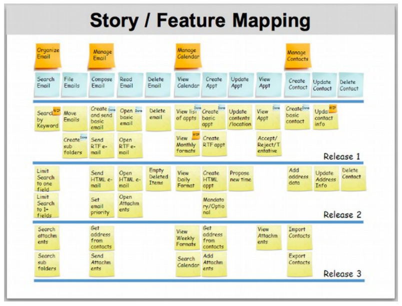
Debrief

Ask a representative from each table to share the group's result with the class. Lead a discussion about how the personas differ from the “roles” and what additional value is obtained by personalizing users in this way.

Mention that personas are usually based on market research, and are not just made up out of thin air. For purposes of the class exercise, it's okay to make up personas out of thin air.

16. Story Mapping / Feature Mapping / Earned Business Value

This section may be skipped at the discretion of the facilitators.



Relevance to Capital One

Despite the great investment in effort and time to carry out release planning and PI planning, teams often receive intent as a large "blob" of "must have" tasks. There is no breakdown based on business value, no clear indication of relative priorities, and insufficient information to determine a minimum viable product (MVP for release.

The agile coaching team has recommended that program teams use two key techniques during release planning to address these issues: Story Mapping (or Feature Mapping) is a method of decomposing desired functionality and identifying subsets of the functionality suitable for incremental releases. Earned Business Value is a method of assigning relative "value points" to each piece of functionality, to help define business priorities for delivery of functionality.

Goals

1. Understand what Story Mapping is for
2. Understand what Earned Business Value is for
3. Understand where these practices are meant to fit in the overall release process
4. Understand how these practices can help development teams relate to the intent

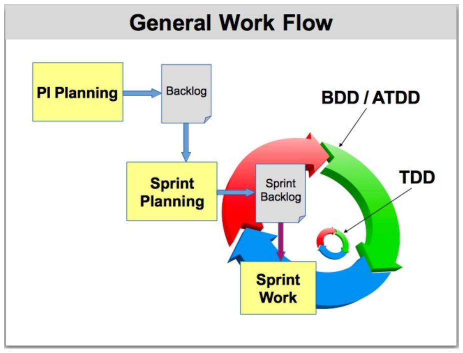
Presentation

Remember that these topics are covered in depth in a separate class. Our purpose here is only to clarify part of the release work flow. Just walk them through the Story Map on the slide and explain what it means.

For EBV, you can explain that business stakeholders would be given a fixed number of value points to play with, and then allocate those points to the boxes on the Story Map. Explain that this is useful for tracking value delivery as opposed to tracking the quantity of work delivered (Velocity). There is no need to go into great depth on these topics.

17. General work flow

This section should not be skipped.



Relevance to Capital One

To many people, it's unclear where various recommended practices fit into the big picture. When teams begin to use practices like ATDD and TDD, they need to know when and where each practice applies. Otherwise, they will spin their wheels trying to do things that don't make sense or failing to do things they ought to do.

Goals

1. Understand the general work flow in a release (high level)
2. In particular, understand where the ATDD and TDD practices fit

Presentation notes

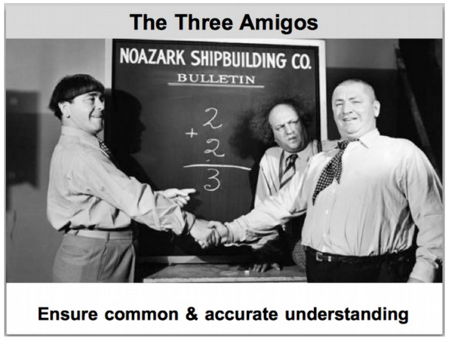
Describe how the ATDD process begins during release planning and PI planning. A few scenarios are defined at that time. As the product backlog is refined to prepare a sprint backlog, scenarios are refined and the suite is expanded to describe the functionality necessary to implement the intent.

Development teams then build the functionality by driving the code from the acceptance scenarios. This is represented by the outer loop in the illustration. To build up the code necessary to support a feature, developers use the TDD technique to drive code from executable unit tests. Unit tests are a lower level of tests than acceptance tests.

Cucumber is used for the outer loop (ATDD). The programming languages in which the application is written are used for the inner loop (TDD).

18. Three Amigos

This section may be skipped at the discretion of the facilitators.



Relevance to Capital One

In the general model of software development that assumes collocated, cross-functional teams, the first step in working on a story is for the analyst, programmer, and tester to touch base briefly to be sure they all have the same understanding of the story.

At Capital One, we don't have feature teams, we mostly have individual silos rather than cross-functional teams, we have no formal analyst role at the team level, and testers generally don't know how to write automation scripts. These realities make it all the more important that different team members who will contribute to a story get together and ensure they all share a common understanding.

Goals

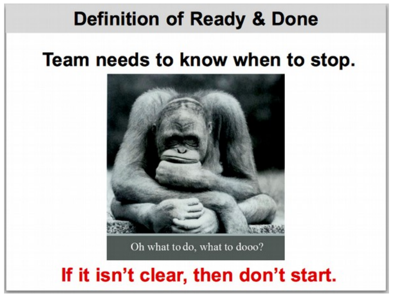
1. Understand how the Three Amigos practice is adapted to Capital One reality
2. Understand why it's important for team members to touch base about a story before starting work in earnest

Presentation notes

Your description of this topic can consist of a rephrasing of the comments in the “relevance” paragraph.

19. Definition of ready and done

This section should not be skipped.


Definition of Ready & Done

Team needs to know when to stop.

Oh what to do, what to dooo?

If it isn't clear, then don't start.

Relevance to Capital One

Most teams accept any and all work requests without question and they begin work on stories without having any clear definition of how they will know when they have finished. To be able to function in the future state environment, teams must have explicit and clear requirements for a story to be workable. Without any definition of the "end" of a story, there is no way to guess how large the story is or how long it will take to deliver.

Goals

1. Understand the purpose of the definition of done
2. Understand the purpose of the definition of ready
3. Understand what to do when the definitions are not met

ACTIVITY 1

Facilitators

Ask participants to self-organize into small groups to work at tables.

Give each table the story cards for the exercise **[TBD]**.

Participants

Discuss each story within your group. For each story, discuss whether it is clear how the team will know when they have completed the work, and why.

Debrief

Each group shares their results with the class.

ACTIVITY 2

Facilitators

Ask each group to examine some of the real stories currently on team backlogs and do the same sort of analysis as they did in the first exercise.

Debrief

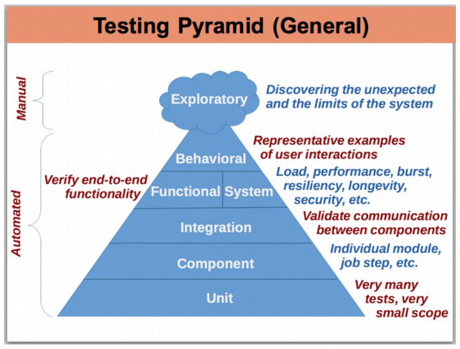
Each group shares their results with the class.

ACTIVITY 3

With the whole class, solicit ideas for items that should be included in the definition of done and the definition of ready for stories.

20. Test automation pyramid

This section should not be skipped.


Testing Pyramid (General)
Manual
Automated
Exploratory
Discovering the unexpected and the limits of the system
Behavioral
Representative examples of user interactions
Verify end-to-end functionality
Functional
System
Load, performance, burst, resiliency, longevity, security, etc.
Integration
Validate communication between components
Component
Individual module, job step, etc.
Unit
Very many tests, very small scope

Relevance to Capital One

Many teams have a fairly imprecise understanding of test automation. Some will say they already have automated tests, but when we look more closely we find they are manually setting up test jobs and executing them on a one-off basis. Some are greatly confused about what a “unit test” is as opposed to a “functional test.” Many people have never thought about code isolation by stubbing or mocking. Many people have an exaggerated fear of the amount of “extra” time will be required to prepare automated tests.

Proper understanding of this topic is a critical success factor for the organizational transformation.

Goals

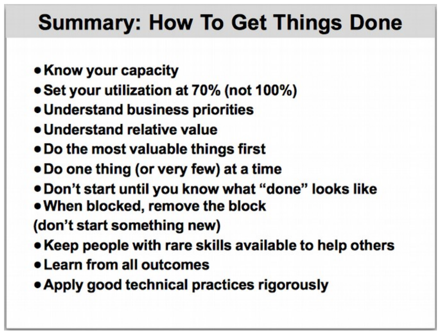
1. Understanding the purpose, frequency, and scope of automated checks at each level of the pyramid.
2. Understanding how the automated scripts are stored in version control alongside the production code.
3. Understand the test-driven procedure for building new code, modifying existing code, and fixing production issues.

Presentation notes

Explain the basics of each level of automated checks. Explain the difference between “checking” and “testing.” Explain how automated checks fit into the larger continuous delivery pipeline. Explain the tools available for each level of automated checking for the technologies used by the participants in this class (will vary from class to class).

21. How to get things done

This section should not be skipped.



Summary: How To Get Things Done

- Know your capacity
- Set your utilization at 70% (not 100%)
- Understand business priorities
- Understand relative value
- Do the most valuable things first
- Do one thing (or very few) at a time
- Don't start until you know what "done" looks like
- When blocked, remove the block (don't start something new)
- Keep people with rare skills available to help others
- Learn from all outcomes
- Apply good technical practices rigorously

Relevance to Capital One

People tend to focus on following rules and procedures rather than on pulling work items all the way through the delivery pipeline to "done." We want people to get into the habit of getting things done, using standard rules and procedures as appropriate but fearlessly bending those rules and adapting those procedures when necessary to get the work done.

Goal

1. Review several key practices that help us get things done.

Presentation notes

Discuss each point and address any questions or concerns from participants.

22. Cucumber and ATDD

This section should not be skipped.

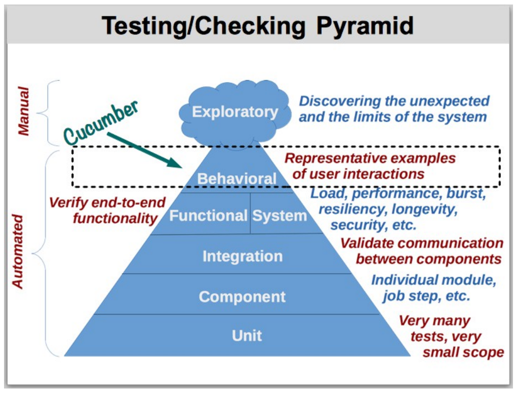
Relevance to Capital One

This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

Goals

1. Understand the terms ATDD, BDD, EDD, Specification by Example, and TDD
2. Understand where Cucumber automation fits in the test automation pyramid

Presentation notes


Testing/Checking Pyramid
Manual
Automated
Cucumber
Exploratory
Discovering the unexpected and the limits of the system
Representative examples of user interactions
Behavioral
Verify end-to-end functionality
Functional
System
Load, performance, burst, resiliency, longevity, security, etc.
Integration
Validate communication between components
Component
Individual module, job step, etc.
Unit
Very many tests, very small scope

Cucumber automation falls at the top of the test automation pyramid. The implications are:

- There are fewer test cases than there are at lower levels of the pyramid
- The test cases have very large scope – end-to-end through the UI

Clarifying terms:

The value of driving functionality from executable examples was recognized and gained popularity in any places around the world at about the same time. Different people invented different terms to describe the practice. The following terms are effectively equivalent:

- BDD – Behavior-Driven Development
- EDD – Example-Driven Development

- ATDD – Acceptance Test-Driven Development
- Storytest Drive Development
- Executable Specifications

Capital One likes the term ATDD.

Note – You will see different interpretations on Pulse, in which people attempt to derive different meanings for these terms based on their subjective understanding of the English words. These interpretations, while well-intentioned, are the work of individuals who have virtually no real experience with these techniques, and who are trying to understand the practices by looking at the buzzwords. You might want to warn participants not to become confused by this sort of information.

23. Gherkin language introduction

This section should not be skipped.

Relevance to Capital One

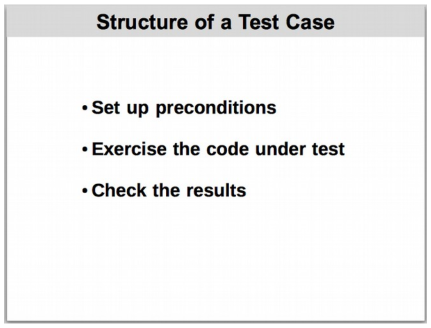
This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

Goals

1. Understand what the Gherkin language is for
2. Understand the relationship between Cucumber and Gherkin
3. See some sample Gherkin and have an opportunity to discuss it

Presentation notes

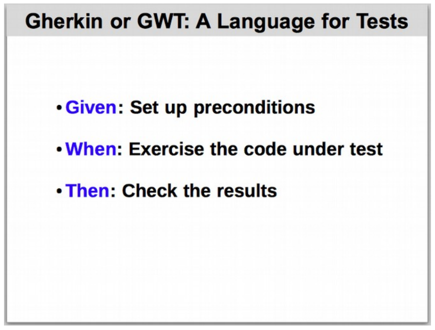
Clarify the common structure of any test case at any level.



Structure of a Test Case

- Set up preconditions
- Exercise the code under test
- Check the results

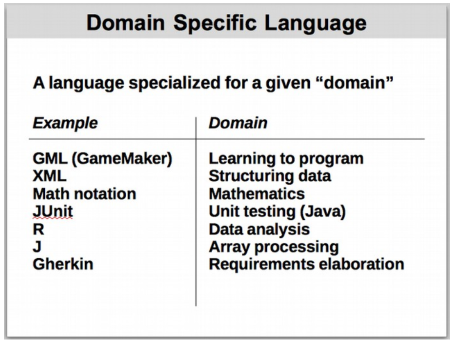
Describe how Gherkin language keywords correspond with the three parts of a test case.



Gherkin or GWT: A Language for Tests

- **Given**: Set up preconditions
- **When**: Exercise the code under test
- **Then**: Check the results

Explain what is meant by *domain specific language* (DSL).

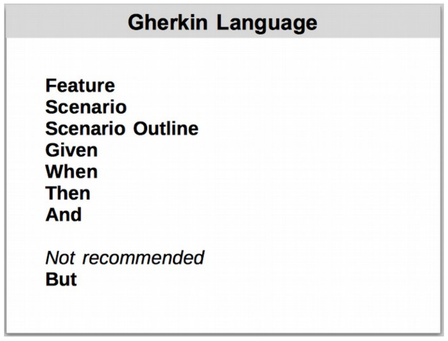


Domain Specific Language

A language specialized for a given "domain"

Example	*Domain*
GML (GameMaker)	Learning to program
XML	Structuring data
Math notation	Mathematics
JUnit	Unit testing (Java)
R	Data analysis
J	Array processing
Gherkin	Requirements elaboration

Explain the keywords of the Gherkin language

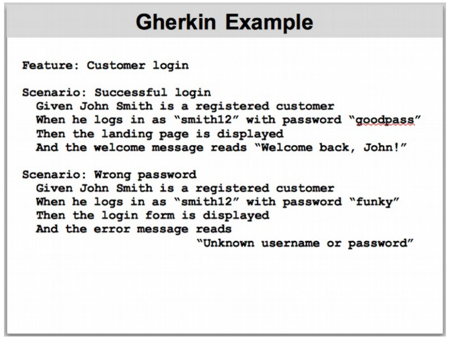


Gherkin Language

Feature
Scenario
Scenario Outline
Given
When
Then
And

Not recommended
But

Walk through an example of Cucumber scenarios written in Gherkin



Gherkin Example

```
Feature: Customer login

Scenario: Successful login
  Given John Smith is a registered customer
  When he logs in as "smith12" with password "goodpass"
  Then the landing page is displayed
  And the welcome message reads "Welcome back, John!"

Scenario: Wrong password
  Given John Smith is a registered customer
  When he logs in as "smith12" with password "funky"
  Then the login form is displayed
  And the error message reads
                         "Unknown username or password"
```

24. Gherkin style and domain language

This section should not be skipped.

Relevance to Capital One

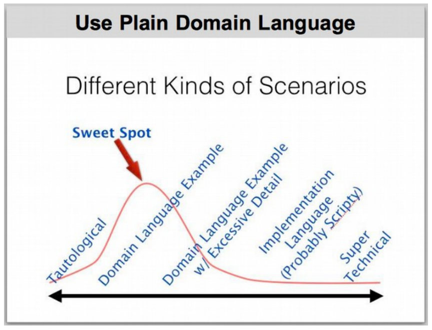
This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

Goals

1. Understand that Gherkin scenarios should be written in domain language
2. Get a sense of the “sweet spot” for writing scenarios in domain language

Presentation notes

Describe the spectrum of Gherkin language styles that commonly occur in organizations that use the tool.


Use Plain Domain Language
Different Kinds of Scenarios
Sweet Spot
Tautological
Domain Language Example
Domain Language Example w/ Excessive Detail
Implementation Language (Probably Scripty)
Super Technical

ACTIVITY

Facilitators

Ask participants to self-organize into small groups to work at tables. Give them the Gherkin scenario cards and categories.

Participants

Discuss each sample Gherkin scenario and categorize it along the spectrum from Super Technical to Tautological.

Facilitators

Some people don't know what “tautological” means. Use an analogy, like “I know pizza is good because it's good.”

25. Gherkin, Cucumber, and ATDD work flow

This section should not be skipped.

Relevance to Capital One

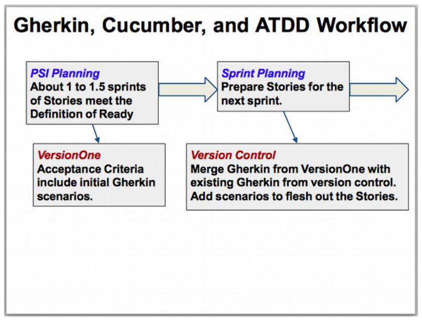
This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

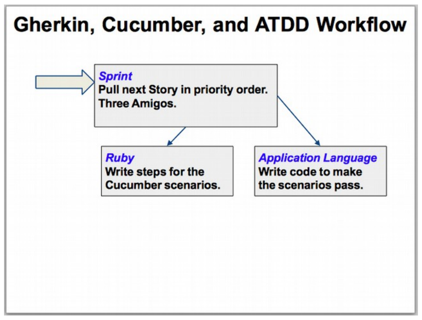
Goals

1. Understand the points in the work flow where teams develop Cucumber scenarios
2. Understand how VersionOne and source version control systems play in the new work flow

Presentation notes

Explain how the scenarios come into existence, how they are used as a discussion tool with stakeholders, and how they are maintained under version control and refined by development teams.


Gherkin, Cucumber, and ATDD Workflow
PSI Planning
About 1 to 1.5 sprints of Stories meet the Definition of Ready
Sprint Planning
Prepare Stories for the next sprint.
VersionOne
Acceptance Criteria include initial Gherkin scenarios.
Version Control
Merge Gherkin from VersionOne with existing Gherkin from version control. Add scenarios to flesh out the Stories.


Gherkin, Cucumber, and ATDD Workflow
Sprint
Pull next Story in priority order. Three Amigos.
Ruby
Write steps for the Cucumber scenarios.
Application Language
Write code to make the scenarios pass.

26. Ruby language introduction

This section should not be skipped.

Relevance to Capital One

This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

Ruby is the language in which Cucumber *steps* are written. The steps are the connecting tissue between the English-like Gherkin scenarios and the application under test. Gherkin alone is useful as a communication tool with stakeholders, but the steps are necessary to automate the test cases.

Goal

1. Get started with basic Ruby language syntax

ACTIVITY

Facilitators

Remind participants that they should have installed Ruby software on their laptops before coming to class.

For some groups, it's necessary to walk through the introduction yourself, projecting your screen. Other groups are self-sufficient enough to work through the introduction on their own. Optionally, encourage people to work in pairs so they can start practicing pairing at the same time as they explore Ruby syntax.

Have participants download this project from Github:

`https://github.com/neopragma/self-start-ruby`

The project is a step-by-step self-guided hands-on introduction to basic Ruby syntax. Have partipants start **`irb`** in a command window on their laptops. This is a REPL (read-evaluate-print-loop) for entering Ruby statements and viewing the output from those statements.

Participants

Using **`irb`**, work through the self-starter project to get a basic idea of Ruby syntax. If you need it, ask a facilitator to show each step on the projector so you can follow along at your own pace.

Facilitators

Toward the end of the exerise, participants will start to type their Ruby code into files instead of using **`irb`**. Many people are confused about where Ruby leaves off and RubyMine (the company-recommended Ruby IDE) begins. If participants want to use RubyMine, they might need considerable hand-holding to get anything to work.

Participants who don't want to use RubyMine may or may not be comfortable using the command line and a text editory. Expect to spend some time assisting participants with this portion of the activity.

27. Cucumber demo and ATDD walkthrough

This section should not be skipped.

Relevance to Capital One

This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

Goals

1. Get a sense of how Cucumber, step files, and application code are related
2. See how a working Cucumber project is set up
3. Experience the work flow of driving a code change from test cases

Presentation notes

Facilitators

Download the project:

`https://github.com/neopragma/hellocuke`

The sample application displays "Hello, World!" in several languages. Show the class how a team would go about completing the following "user stories:"

1. Changing the phrase that is displayed for one of the languages that is already supported
2. Adding a greeting for a new languages

Key points to convey:

1. We are going to follow the standard TDD cycle: Red-green-refactor.
2. Complete one story at a time. Don't try to multitask both stories.
3. First run the cukes to be sure all the scenarios are green. If not, it means someone changed the application without modifying the test cases (naughty). Your team wants to begin any code changes with a clean slate – no broken tests.
4. Change the **`.feature`** file so that one of the scenarios expects different texts. Run the cukes. The scenario you changed should now fail while all the other scenarios still pass. This is the *red* step in the red-green-refactor cycle: "Red for the right reason."
5. Change the application so that it displays the expected text. (This amounts to changing the text

in an HTML SELECT element.)

6. Run the cukes again. This time, all scenarios should pass. This is the *green* step in the red-green-refactor cycle: “Write the minimum amount of text necessary to make the test pass.”
7. Now mention that the team would do the refactor step of the red-green-refactor cycle next. In this case, there is no duplicate code and no other “code smells” to deal with. All we did was modify a little bit of text. The structure of the application is unchanged. But we still take the time to examine the code and *explicitly decide* that no refactoring is needed.
8. Ask participants if they are ready to accept the story. (It's okay to accept it.)

The second story is to add a new language to the application. This will introduce a new wrinkle in development.

Your walkthrough will be mostly the same as for the first story. The difference is that there is a helper method (in **`helpers.rb`**) that has to be changed so the application can look up the name of the new language. You should “forget” to change the helper method so that the new scenario will still fail even after you've made the same sort of change to the application.

28. Writing Cucumber steps

This section should not be skipped.

Relevance to Capital One

This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

Goal

1. Practice writing steps to automate Cucumber scenarios.

Presentation notes

Facilitators

At this point insert an appropriate tutorial project using the technologies relevant to the participants in this class (mobile, web, mainframe).

29. Test-driven development

This section should not be skipped.

Relevance to Capital One

This is core content for the training course, covering a key practice necessary to achieve the organizational goals of continuous delivery and improved software quality.

Goal

1. Experience the TDD work flow at the unit level.

Presentation notes

Facilitators

Use the well-known FizzBuzz exercise as a vehicle to introduce TDD. Several starter projects are available on Capital One's internal Github repository:

Cobol
http://github.kdc.capitalone.com/nbn240/cobol-fizzbuzz

Java
http://github.kdc.capitalone.com/nbn240/java-fizzbuzz

Python
http://github.kdc.capitalone.com/nbn240/python-fizzbuzz

Ruby
http://github.kdc.capitalone.com/nbn240/ruby-fizzbuzz

Options

- Depending on the current level of knowledge of the class, you might have to project the project and walk them through the first few steps.
- Incorporate pair programming with TDD – have participants work in pairs on the exercise.
- Randori style – have a pair come to the front of the room and project their work. Switch out one pair member after each red-green-refactor cycle or on a time-boxed basis.
- If you have a mixed group, different participants may work in different programming languages.

30. Supplemental material for mainframe audiences

This section may be skipped at the discretion of the facilitators.

Relevance to Capital One

This material is relevant to development teams who develop and support applications that run on the IBM zOS platform.

Goals

1. Understand how the various levels of automated testing depicted in the testing pyramid correspond with mainframe technologies
2. Understand appropriate code isolation to support each level of test automation
3. Understand which tools are available to support test isolation at each level

Presentation notes

Use the supplementary deck that presents information for mainframe audiences.

Hand out the sample JCL for component-level testing of batch job steps. **[TBD]**

Mainframe teams typically have many questions about how these practices can work in their context. Be prepared to field specific, in-depth, real-world questions with practical, realistic answers. There must be at least one instructor who has direct professional experience on the mainframe platform when presenting the class to mainframe development teams.

31. Tie-back to participants' goals

This section should not be skipped.

Goals

Ensure we have provided participants with the information they need

Presentation notes

Revisit the expectations and desires the participants listed at the start of the class and make sure each topic has been addressed to their satisfaction. If it is not practical to address a topic adequately in a few minutes, then schedule a follow-up with interested participants to help them with the topic in question.

32. Retrospective

This section should not be skipped.

Goal

Solicit feedback from participants to help us improve the training.

Presentation notes

Use any retrospective format you like. Collect feedback and share it with other instructors and with the Capital One training department.