I’m not a bottleneck!
I’m a free man!

Playing to apply the Theory of Constraints and the “5 Focusing Steps” for process improvement

Pascal Van Cauwenberghe
pascal@nayima.be
Nayima
http://www.nayima.be

Portia Tung
portia@portiatung.org
Exoftware
http://www.portiatung.org

Table of Contents

Why and when to use this session

Format and length

1. Tutorial + simulation game to introduce concepts: 60-90 minutes
2. Workshop to apply the concepts to real-world cases: 60-90 minutes

Why and When

Use this session to:

- Get an overview of the way a team or organisation works
- Decide where to apply improvement effort
- Help team members look beyond their own team, experience a process from the perspective of different participants and observe how different teams collaborate (or not) during delivery
- Let participants in a process see for themselves how the process looks from the customer's point-of-view
- Involve participants in the identification of issues and improvement ideas for their work
- Introduce the Theory of Constraints, Lean and Agile process improvement concepts
- Analyse a business process that is about to be automated, as an initial step in requirements discovery.

This session is typically used at the start of a process improvement effort in order to get an overview of the existing process as well as identify where best to apply improvements. Use this session when you want to widen the scope of an improvement effort so that everyone becomes aware of the "big picture" (strategic) approach.

Participants

Ensure you include everybody who will be affected by the process improvement, namely the people in the team affected and those in teams who interact with that team.

No previous knowledge of the Theory of Constraints, Lean or Agile is required. Some familiarity with development processes and the current way of working is required to apply the techniques to real-world situations.

Participants will have one of two roles:

- Player in the simulation. There are 7 roles to choose from
- Observer/consultant. Observes the players during the simulation and offers advice for process improvement after every simulation round.

The participants can be split into 2 (or more) teams, each with a set of players and group of consultants. Each team requires a session coach. Ensure that you perform debriefings and improvement discussions plenary so that all participants share the full learning.

Maximum number of participants: +/- 20 (7 players + 13 observers) per coach.

Coaches

At least one coach per simulation team to introduce the theory, manage the simulation, moderate debriefing and discussion and provide hints and tips.

The coach should be familiar with the Theory of Constraints, Lean and Agile and have experience of applying them in real projects. And, of course, they should have basic coaching and session leading skills and experience.

Prepare for the session

Prepare the session materials

Print out one Job Instruction Sheet per simulation team. Print out enough handouts to distribute after the session or let participants know where they can download the handout from.

Have folding papers and writing materials ready to distribute at the start of the simulation.

Lay out the room

Simulation

Arrange the players in a "production line". Each player sits on a chair along one side of a single row of tables. All players should sit on the same side of the row of tables. The simulated "work" will be done on the row of tables.

The observers sit on the other side of the table a little distance away, so that they can comfortably see what the players do without interfering with the production line during production.

If there is more than one simulation team, place each row of tables along a wall of the room with players facing towards the middle of the room. Observers sit in the middle of the room, so that they can watch one or more simulations.

Workshop

Participants work in groups of 4-7 people. Each group needs table space and chairs. Provide them with plenty of workshop materials.

Prepare the participants

Invite the participants and let them know what they can expect: a fun, playful and safe way to learn about and experiment with techniques from the Theory of Constraints, Lean and Agile.

If you also run the workshop to apply the techniques on real-world situations, make sure that the goal of the session is very clear: to understand how the *whole* process works and where and how we can make improvements.

Take care that you keep the session "safe": being the bottleneck doesn't mean you (or your team) did something wrong! The goal of the tutorial and simulation is to show that **there is always a bottleneck** and that this is not a problem or a reason for blame.

If the concepts of the tutorial are not well understood or if the participants do not trust each other, do not run the workshop. Identify and fix the problem at root cause first.

Short session description to tailor for your invitation

The "Theory of Constraints" states that the throughput (value of useful output) of a system depends on exactly one constraint (or bottleneck). If we wish to improve our

system, we need to improve the bottleneck. If we wish to strengthen a chain, we need to strengthen the weakest link.

How do we find the bottleneck? Once we've found it, what can we do about it?

In this session, you'll learn all about the Theory of Constraints and the "5 focusing steps" for process improvement by:

- Participating in a simulation where you can experiment with the different techniques
- Applying your newly learned skills to improve some real-world processes. Or better still: have your process improved by the participants of the session.

In the first part of the session, you participate in a simulation of a small company. Each of the elements of the Theory of Constraints and the 5 focusing steps for process improvement are introduced and applied in the simulation. At the end of the simulation you will have had fun and experienced all the necessary techniques. By the end of the game, you become a Theory of Constraints "consultant".

In the second part of the session, you can come forward to play the "Customer" role. You explain your situation and what you want to achieve; the "Consultants" will help you to optimise your system using the techniques they just learned.

You should return to work with lots of useful ideas for process improvement; you understand better why Agile works; you will see why you need to look at the whole system, the whole value stream and not just software development.

You may even go home with a complete management briefing to improve your company.

Run the tutorial + simulation – Round 1

Introduce the session and set up the simulation

Goal of the session

Play the game to:

- Learn how to visualise processes and their goal
- Learn how to know where and how we should improve the process
- Learn how to apply Lean and Agile techniques to improve the process.

Acceptance Test: after attending this session, you should be able to apply the Theory of Constraints, Lean and Agile techniques from the session to your work, your organisation and your work process.

Create the simulation team(s)

7 volunteers are required to play the 7 roles of the team in the simulation. The simulation runs for 3 rounds of 5 minutes each, followed by a debrief. A player may step out of the simulation whenever they want. No special skills or knowledge are required. Everything that is needed to participate will be taught at the beginning of the simulation round.

The other participants act as observers/consultants. During the simulation, they observe what happens. After each round they report what they have observed and can propose improvements to the players in the team(s). If a player steps out of the simulation, another volunteer can take their place.

Job Instruction

Team goals:

- Create as many pairs of boats and hats as possible (note that individual boats and hats are worth nothing)
- Use as few sheets of paper as possible (minimise waste).

You can give out chocolate or candy at each round (wages) and give the team bonus candy or chocolate per accepted pair of boats and hats (profit sharing).

Provide each player in turn with their Job Instruction Sheet, writing materials and folding paper they need to perform their ‘work’.

For each player:

- The coach explains the task in the Job Instruction Sheet to the player and observers
- The coach demonstrates the work to be performed
- The player tries out the work to be performed
- The coach verifies that the work was performed satisfactorily and that the player has understood the work. If not, explain, demonstrate and try out again. If someone feels unable to perform the job, they may leave the simulation and be replaced or swap roles with another player.

The different roles are:

- Customer Requirements: Tells the team what the customer needs; gives folding paper to the team; counts how much paper was given to (invested in) the project. Requires folding paper and a pen
- Analyst: Communicates requirements from the customer; performs the initial folding. Requires a pen or pencil
- Designer: Continues folding
- Programmer: Finishes folding. If the customer asks for a hat, there is little to do. To fold a boat, first make a hat then perform the extra steps required to fold a boat. Hint: Note the Programmer job instruction sheet is two pages long
- User Interface Designer: Decorates a hat with *exactly* one flower on each side
- Decorates the boat with *exactly* three portholes on each side of the boat and *exactly* one anchor on one side of the boat. Requires several colouring pens or pencils
- Tester: Verifies the product of the team against the acceptance criteria on the Job Instruction Sheet. The Job Instruction Sheet defines what to do when a fault is found: return the product to the person responsible for the fault
- Customer Acceptance: Perform final acceptance check against the written acceptance criteria. Count the number of finished pairs of boats and hats. Requires a pen.

Run the first round

Make sure everybody understands the goal of the simulation, the rules and their job responsibilities.

The simulated "work day" lasts for 5 minutes. Make sure that the players and observers are aware of the time passing. The short amount of time puts pressure on the team to work quickly and concentrate on the job at hand. They don't have the time to over-analyse what they do. Some ways to increase the pressure are:

- Use a big hourglass, so that time passing is visible to all
- Use an egg timer with a loud ring, so that the end of the round can be heard loud and clear
- Use a stopwatch (think Taylor) and call out the time. Increase the call out frequency gradually: "*4 minutes to go, 3 minutes to go, 2 minutes to go, 90 seconds to go, 60 seconds to go, 30 seconds to go, 15 seconds to go, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1. Time's up!*"

Ensure that observers only observe. They should remain silent and not interfere with the players. Verify that the players follow the rules and that Customer representatives count the input and output of the simulation.

Pay out to the players at the end of the round: one chocolate/piece of candy per player per round + 1 chocolate/piece of candy per team for each accepted pair of boats and hats.

Stop the simulation.

Debrief the first round

If everything goes according to plan you should see:

- It takes a while to fill the pipeline. Meanwhile, the people at the end of the production line are idle most of the time
- Paper starts to pile up in front of the Programmer

- All players up to and including the Programmer are busy
- The players after the Programmer have idle periods
- Faulty work detected by the Tester is sent back down the line and disrupts the affected player by requiring time be spent on correcting (often fundamental) problems
- There is a lot of paper inside the system. Most of it is waiting to be folded by the Programmer.

Ask the observers what they saw. They should have at least seen the above points. Stick to observations and facts; leave analysis and improvements for discussion later.

Ask the players how they felt. How did the busy people feel? How did the idle people feel?

Introduce the 5 focusing steps

The Theory of Constraints has a five step program for optimising systems. Each step is named and introduced by the coach(es). The coach writes each step on the flipchart/whiteboard. Some examples are given. The observers can now apply each step to the simulation. The players in the team choose one example of each of the three types of optimisation techniques (Exploit, Subordinate and Elevate) to apply in the next round.

Step 0: What is the goal?

Each system has a goal. What is it you want to achieve? How will you know when you have reached the goal? What is the acceptance test? How will you know that you are getting nearer to the goal? What is your management metric?

This is the most difficult step in the whole process. It may take longer than expected to identify. What is the goal of your company? What is the goal of your team? What is your goal? How do you know you're moving towards the goal?

Ask five people in your company this question and you are likely to get 5 different answers. What's more you will probably get conflicting answers. If you can't agree on the goal(s), there can be no improvement.

Ask participants what the goal of the simulated system is. You asked them at the start of the simulation to:

- Create as many pairs of boats and hats as possible (note that individual boats and hats are worth nothing)
- Use as few paper sheets as possible.

Beware! The prioritisation of the goals is important. Optimising the use of paper is easy: do nothing. This means you will get no income.

Ask the Customer representatives to announce how many pairs have been accepted and how much paper was used. Write the scores on another whiteboard/flipchart in a grid, so that we can see the evolution of those two numbers over the rounds.

Explain that Throughput Accounting uses a very simple management metrics system with three variables:

- **Throughput**: Measures how much of the goal you realise. Usually you measure how much customers pay you, but there are alternatives for non-profits. In the simulation, this is the number of pairs accepted (and presumably paid) by the customer minus the profit sharing chocolates/candy
- **Investment**: Measures how much money we tie up providing the means for production. In software terms this includes machines, knowledge, requirements. In the simulation, this is the number of paper sheets inside the system
- **Operating Expense**: Measures how much money we spend to keep the system running. For example: rent, electricity, heating, wages. In the simulation this is 7 chocolates/pieces of candy per round.

There are two simple formulae to relate these variables to company goals:

Net Profit = Throughput – Operating Expense

Return on Investment = Net Profit / Investment

To improve, we first try to increase Throughput, then try to decrease Investment and then (as a last resort) try to reduce Operating Expense. Most "optimisation" or "waste elimination" initiatives make the mistake of tackling Operating Expense first, with not surprisingly little effect. It's easy to reduce costs at the expense of throughput and profit.

Step 1: Identify the bottleneck

Each system has one constraint (one person, one team, one machine, one rule) that determines the throughput of the whole system. The bottleneck is where the constraint manifests itself.

A chain is as strong as its weakest link. If you want to make the chain stronger, strengthen the weakest link. This is the essence of the Theory of Constraints: find the bottleneck and do something about the constraint to improve the system as a whole. Make improvements anywhere but the bottleneck and your efforts will have no effect or, worse still, create a negative effect.

How do you recognise a bottleneck?

- They are very busy, all the time
- Work piles up in front of them
- People downstream of the bottleneck are idle some of the time.

Ask the participants where the bottleneck is in the simulation. The symptoms should be easy to recognize.

By design, the Programmer is the bottleneck. This does not say anything about the role, the person performing that role or their skill! Each system has a bottleneck. The bottleneck is the most important place in the system.

Beware the bottleneck complex: Being the bottleneck is not bad, although some people hate being the focus of attention. Conversely, being the bottleneck is not good, although some people like the attention.

Now that we know the bottleneck, what can we do about it?

Step 2: Exploit the bottleneck

We first try to "exploit" the bottleneck. "Exploiting" is the technical term for ensuring that the bottleneck is not distracted by non-throughput producing work. Because the output of the system is determined by the output of the bottleneck, any waste at the bottleneck results in less output for the system.

How can we exploit the bottleneck?

- Ensure that the bottleneck works on only one task at a time. Task switching is a form of waste
- Ensure that the bottleneck always works on the highest priority, highest value work. More value from the bottleneck means more value for the system
- Ensure that there is always something *useful* for the bottleneck to work on. An idle bottleneck directly reduces system output
- Take away any non-throughput generating work from the bottleneck. What brings more value to the system: producing work or filling in timesheets?
- Eliminate any wasted effort: make sure all the materials and information the bottleneck needs are readily available. There should be no need for the bottleneck to go get (or, worse still, *hunt*) for them

- Minimise the time that a bottleneck has to wait; that waiting time could have been used to generate value. For instance, a bottleneck should get the fastest computer or the best material possible.

Warning: making the bottleneck work for longer is an obvious way to exploit it. Use this exploit with care because you don't want to burn out the bottleneck.

Ask the participants to come up with their own 'exploits' for the simulation. Some examples include:

- Ensure that there is always at least one paper ready to be folded by the Programmer
- When there are issues with the folding, the Tester can send the faulty items back to the Designer instead of the Programmer. The Programmer is not interrupted and can concentrate on producing new items. Of course, this means the Programmer no longer gets feedback on their faulty work
- The Designer consistently puts their output in the most convenient place and in the most convenient orientation, so that the Programmer can take the paper easily and get to work without extra movement or thinking.

When you improve a real system, select the most promising Exploit improvement, implement it, measure the result and go back to step 0. You should first implement all Exploit improvements before trying any other improvements.

Exploiting is a simple improvement because

- It only affects the bottleneck, one person, one team, one machine… You don't have to involve a lot of people in the improvement
- Exploiting is essentially "free": there's no need for extra investment or operational expense, you're getting more value from the resources you have already paid for.

In the simulation, ask each team to select one Exploit and write it down on the game flipchart/whiteboard. Move on to the next step.

Step 3: Subordinate every decision to the bottleneck

Now the bottleneck has been identified and exploited what next? You "Subordinate" every decision to the bottleneck. The bottleneck is the most important part of the system. To subordinate to the bottleneck, the rest of the system must work to help the bottleneck produce the maximum amount of value.

How can we subordinate decisions to the bottleneck?

- Those who provide input to the bottleneck ensure that the work received by the bottleneck is of the highest quality, so that the bottleneck does not waste time with identifying and dealing with issues. Provide the work to the bottleneck in a way that is convenient to the bottleneck
- Those who work with the output of the bottleneck must ensure that they don't waste any of the bottleneck's work by introducing issues or spoiling the work altogether
- Let everyone work at the rhythm of the bottleneck, no faster, no slower. If those who provide input to the bottleneck work faster, they will only increase the pile of work in progress in front of the bottleneck. Create a small buffer of work for the bottleneck, and ensure that it is always filled with a small, fixed amount of work, so that there's always work to do for the bottleneck and there

isn't too much work in progress. This is called the "drum-buffer-rope" technique

- By definition, those who are not the bottleneck can work faster than the bottleneck. If they work at the speed of the bottleneck, they will have some idle time. They can use this idle time to help and support the bottleneck, to improve the work area and processes, to do research. They can work on other projects, but bottleneck-related work must always take precedence. When the bottleneck is exploited, they have little slack. **Subordinated resources should always have slack**, otherwise they might starve the bottleneck or might not be able to consume all the bottleneck's work when fluctuations arise
- Let another resource take over some work from the bottleneck. This is difficult if the tasks are very specialized, which is why Agile and Lean like to work with generalist-specialist or multi-skilled workers.

Ask the participants to come up with their own 'subordinations' for the simulation. Some examples include:

- The Designer ensures that the buffer before the Programmer is always filled with one item
- All the players before the Programmer work at the rhythm of the Programmer: the Designer stops working when there is one sheet of paper in the buffer and they have another sheet of paper ready. The Analyst starts to fold the next sheet of paper when the Designer takes the one that is ready. Similarly, the Customer representative gives a new sheet of paper to the Analyst when the Analyst gives a folded sheet to the Designer. This "pull" system reduces the waste of paper and slows down the players before the Programmer. They can use the extra time to ensure that their work is done well
- The Designer can take over some of the work of the Programmer. For example, it's easy to complete the hats, you only need to turn down the corners
- The Tester can share the acceptance criteria with the rest of the team
- The Tester can use their slack time to test sooner as well as test the quality of the work before it goes into the bottleneck and immediately after it comes out of the bottleneck.

One way to subordinate to the bottleneck is by enlarging the skillset of the subordinating resources during their slack time to take over some work from the bottleneck. To do this, participants can develop an existing skill or acquire a new skill to enable them to do some or all of the work of another role in the next round. The actual learning of how to fold happens in between rounds. Where the cost is negligible, the skill can be applied immediately to subordinate to the bottleneck.

Role	**Skill**	**Cost**
Designer	Learn to fold hats	Negligible as only requires learning additional step of folding corners
User Interface Designer	Learn to fix cosmetic bugs (eg straightening edges / minor tweaking NOT refolding of entire hat or	Negligible as only requires making of cosmetic changes

	boat)	
User Interface Designer	Learn to fold hats	Throw a die to roll a 6
Analyst	Learn to fold hats	Throw a die to roll a 6
Designer	Learn to fold boats	Throw a die to roll a 6
User Interface Designer	Learn to fold boats	Throw a die to roll two 6s in total

When you improve a real system, select the most promising Subordination improvement, implement it, measure the result and go back to step 0. You should first implement all Subordination improvements before trying any other improvements because:

- Subordinating is more difficult than exploiting as it involves more of the system
- Subordinating is essentially "free": there's no need for extra investment or operational expense, you're getting more value from the resources you have already paid for.

In the simulation, ask each team to select one Subordinate and write it down on the game flipchart/whiteboard. Move on to the next step.

Step 4: Elevate the bottleneck

When you can't find any more Exploits or Subordinations, you should try to "Elevate" the performance of the bottleneck. You Elevate by investing time and money to improve the bottleneck's performance.

This is the improvement most people intuitively jump to. Of course, it's not that easy.

How can we elevate the bottleneck?

- Get more of the resource: more people, more machines and faster machines
- Give people training and better tools
- Provide coaching and mentoring
- Hold team retrospectives and turn "what went wrong" and puzzles into actions
- Hold one-to-one meetings with team members for individual improvement
- Improve the workspace
- Improve the process
- Pairing to catch mistakes sooner and to learn from each other.

Ask the participants to come up with their own 'elevations' for the simulation. Some examples include:

- Add another Programmer. That costs an extra piece of candy/chocolate per round
- Replace the Programmer by another participant who's better at folding. That costs some training time to explain to the new player how the simulation works
- Provide training for the Programmer by letting another session participant teach the Programmer to fold. That costs an extra piece of candy/chocolate for the trainer
- At the moment we're spending time to do a retrospective and improve the process. That costs time and we are not producing value, but we are investing in the future.

When you improve a real system Elevate as a last resort, when you can't find any more Exploit or Subordinate improvements because:

- Elevating costs time and money. These changes are no longer free
- At first, the change will have negative effects as the new people are integrated and new techniques are assimilated.

Let the team choose an elevating improvement and write it down on the team board. Don't allow the team to elevate by adding people in the second round: the effect is too great and will obscure the effect of other improvements. If the team asks for another Programmer, tell them you will go and recruit one but that they will not be available for the second round.

Step 5: And again!

Each time you find an improvement, implement it, measure the results and go back to step 0. Re-examine if the goal is still valid.

Always re-identify the bottleneck. Improving the system will often make the bottleneck move. In the simulation, the bottleneck is likely to move to the Designer if they take on some of the work of the Programmer and the Programmer is elevated. If the bottleneck moves, stop improving the old bottleneck and start improving the new bottleneck.

Keep improving, there's always a way to do better. Use the knowledge, experience and creativity of *everybody* who's involved in the system. Don't let inertia become the constraint!

Run the tutorial + simulation – Round 2

Run the second round

Each team has selected three improvements, one of each type. In this round they can implement these (and only these) three improvements. In real projects we would only apply one improvement at a time, so that we can measure the effect and do not risk interference between multiple improvements. In the simulation, we want to speed up improvements but don't want to change too much at once.

If a player wishes to step out of the simulation, another participant may take their place. The player who leaves should train their replacement. If a participant provides training to a player (a possible elevation), they get paid one piece of candy/chocolate.

The game is reset: all work in progress is removed from the playing table. Run the game again for 5 minutes.

Debrief + improve the second round

Record the throughput and investment for each team. Compare the results of the teams, if there is more than one team. Most of the time throughput does not improve much in the second round, but investment goes down because the team implements a pull system and works at the pace of the bottleneck. Because the non-bottleneck players now work at the pace of the bottleneck, they have more time to observe, think and implement improvements. We expect to see quality go up as players take more care with their work. Throughput will go up slightly because the players now have more experience.

Take the teams through the 5 focusing steps again. The goal has not changed, but the bottleneck may have moved: the Designer often becomes the bottleneck. Select one improvement of each type per team.

Ask the participants what would happen if we improved a non-bottleneck. If we improved someone downstream from the bottleneck, we don't increase throughput, we increase their idle time. If we improved someone upstream from the bottleneck more work in progress would pile up in front of the bottleneck. If we improve a non-bottleneck we might get no result or make the result worse!

Optional: introduce "Real Options" to exploit the bottleneck

The Customer will ask the team to make a hat, a boat, a hat, a boat… This is the perfect plan to produce a set of hats and boats. Unfortunately, the team is not perfect. The team might end up with an unbalanced number of boats and hats because hats are more difficult and are more likely to contain bugs. As the customer only pays for pairs of boats and hats, the surplus hats are waste. The effort of the bottleneck in making these hats is wasted!

Most teams encode which product to make by using the colour of the paper. This is very convenient for team communication, but it means the decision is taken very early: when the customer representative injects the paper into the team. The customer responsible for accepting (and the Tester) knows better if boats or hats should be made: whatever there is a shortage of. The decision to make a boat or hat only needs

to be made when the Programmer starts to work. Until then, all the work is identical for boats and hats.

To exploit the Programmer better, the Customer Acceptance role or the Tester should determine what to build, based on the boats and hats that have been produced and those that are in progress from the User Interface Designer on. All the roles before the Programmer do not need to know what to make. The Programmer needs to be told what to make. The Designer can subordinate to the Programmer by being told what to make and making it convenient for the Programmer to know what to make. If the Designer has subordinated to the Programmer by finishing hats, this is easy: the Programmer only gets the boats. Otherwise, the Designer can place boats and hats in one of two slots, where the Programmer picks them up.

This change uses a "Real Options" approach:

- The decision is taken as late as possible, the option is kept "open" as long as possible: we decide what to make when the Programmer works on the paper, not when the paper is injected in the team
- Meanwhile, more information is gathered: the team looks at finished and in progress work to determine what to make
- The cost of this option is quite low: we change who takes the decision to build a boat or hat. The Tester and Customer spend a bit more time looking at finished work and work in progress, but that's not a problem: they are not the bottleneck and have idle time.

When you have gone through the 5 focusing steps, introduce the sixth focusing step.

Step 6: Change the system

The pursuit of quality demands we improve continuously. This means that when we've improved an existing system as much as we possibly can, it's time for a change. We change the system itself so that we can continue making improvements.

Changing from one system to another is much harder than making incremental improvements to an existing system. To succeed, we need to understand why people resist change so that we can facilitate the system change:

- "*It's not my fault, it's not my problem*" – Achieve consensus on the core problem to get buy-in: "a problem shared is a problem halved"
- "*Your solution doesn't solve my problem*" – Ensure the solution solves the problem for the majority of stakeholders to address their question of "What's in it for me?"

Habits form almost immediately once a system is in place. The result: a mental model made up of assumptions that colours our sense of reality:

- "Hidden rules" that are not explicit, but still followed: "*That's the way we've always done things around here!*" Why did we decide to do it that way?
- "*You can't touch that! You'll break everything!*" How/where will it break?
- "*You don't understand!*" Can you explain it to me?

To see our way out of the problem, we need to invalidate our assumptions:

- Identify the problem
- Identify conflicts that prevent us from solving the problem
- Consider the assumptions that give rise to the conflicts
- Invalidate each assumption (this is called an 'injection')
- Apply each injection to bring us closer towards our solution.

Ask the participants to identify the key problem that's stopping them from changing the system. Challenge their assumptions and encourage them to invalidate their assumptions one by one by applying each injection to the existing system to create a new system.

Run the tutorial + simulation – Round 3 and further

Again, 3 improvements are selected and implemented. Reset the game. Run the simulation for 5 minutes. Record the results. Introduce the seventh step.

Step 7: Design the system. Choose your bottleneck

When you design or change a system, the most important decision to take is "where shall I place the bottleneck?"

A Lean system will typically make the customer the bottleneck: the whole system works at the pace of the customer buying, levelled out using "Heijunka". This pace is called the "takt time".

Remember the bottleneck complex? Being the bottleneck is not bad, although some people hate being the focus of attention. Being the bottleneck is not good, although some people like the attention.

One of the key goals of the Theory of Constraints is to improve predictability by driving out variation. That's why it's important to always know where the bottleneck is. The most important policy a company needs to define is where to *put* the bottleneck. A bottleneck helps establish control and focus.

In the case of software delivery projects, during the first pass of system improvement, the bottleneck is typically the development team. As the development team improves the way they work in relation to the system as whole, the bottleneck can shift upstream (writing stories, defining and starting projects) or downstream (testing, installing, accepting or rolling out new releases).

You should keep the bottleneck within your team if you don't have control of the improvement of the whole system. In an ideal situation, you would have control to make improvements across the whole organisation. For instance, if you were tasked with improving the development team, then you would typically make the team the bottleneck so that you have sufficient control to keep making improvements.

To keep control of the bottleneck:

1. Pick a bottleneck.
2. Identify potential bottlenecks in the near future.
3. For each potential bottleneck, apply the 5 focusing steps just in time before they become the bottleneck.

Run the workshop

Introduce the session and create workgroups

Goal of the session

Apply the techniques learned in the simulation, acting as "Theory of Constraints Consultants", to real systems brought to the session by participants who act as "Customers" to:

- Understand a real system
- Identify the bottleneck in the system
- Identify at least one exploit, one subordination and one elevation improvement action.

Acceptance test: the "Customer" has at least 2 concrete exploiting and/or subordinating actions that they will apply when they're back at work.

It is possible to run this workshop without running the simulation first. Introduce the theory of the focusing steps as you go through each step in the workshop. Participants will understand the theory better if they've seen and felt it in action in the simulation.

Explain the goal of the session

Now that participants know about Theory of Constraints in terms of Lean and Agile, they can apply this knowledge to real cases brought to the session by participants. The goal is to help the owner of the case (the "Customer") to identify actions they can take to improve their system.

There are some basic rules that must be observed:

- All information received from the customer is to be treated confidentially. If the customer has any doubt about whether they can disclose information, they should not present their case to the group
- The customer can decline to answer a question from the other participants
- The customer genuinely wants to understand and improve their system
- Consultants should only ask questions to genuinely understand their customer's system, not to push some agenda or solution. If participants lack questioning skills, let them exercise using the "9 Boxes" interview technique first
- The session should not be used to find out who's "wrong" or who's "the cause of the problem". As any system always includes a bottleneck, no individual or team should be blamed or looked down on because they are (part of) the bottleneck.

If any individual or groups act against these rules, stop the simulation, remind participants of the rules and ask them to commit to following the rules.

Create workgroups

Participants who bring a system to the session come forward and briefly explain what their system is about. They will act as "Customers".

The other participants act as "Consultants" and choose which Customer they would like to help by applying their Theory of Constraints knowledge. Ideally, form groups of about 1 Customer per 3-5 Consultants.

Each workgroup sits at a table with workshop materials: a few sheets of flipchart paper, pens, pencils, post-its…

The rest of the session is run as a series of short tasks to keep participants focused and to avoid analysing too deeply. Often the Consultants will try to come up with solutions too fast and skip some of the steps. The session coaches should make each group stick to the focusing steps. The focusing steps are there to ensure that we do a thorough analysis and not jump to conclusions.

The session coaches are available to help and answer any questions the groups have. The coaches should go to each of the teams, to see if they're stuck and to offer their help.

Step 1: The system and its goal (5 min)

The Customer describes the system that they want to examine. This can be a team, a group of teams or a whole organisation. Make sure that everybody understands the boundaries of the system: what's included in the system and what is not?

The easiest way to discover the goal of the system is to start with the customer of the system: who uses the outputs of the system; who pays for the output of the system? Even if the system is only concerned with an internal team that doesn't directly interact with the external paying customer, it is best to start with the external customer. Reason backwards from the customer's goals: what needs to happen to reach the customer's goal? This is a new sub-goal. What needs to happen to reach that sub-goal? Continue until you arrive at a sub-goal that is achieved by the system you examine. If you discover that the system doesn't contribute to the customer's goal, examine whose goals the system serves.

Throughput is the way that is used to measure that the system reaches its goal. Define the throughput measure for the system. Any improvement will have to improve that metric.

Example from the simulation:

- System = Producing hats and boats from Customer request to Customer acceptance
- Goal = Create as many as possible hat+boat pairs, using as little paper as possible
- Measurement =
    - Throughput = pairs of hat+boat (or what the customer pays for each pair) – profit sharing candies/chocolates
    - Investment = paper injected into the system

Step 2: Visualize the system (5 min)

Now that you know the system and its goal, visualize the system components. Start with the "customer" of the system receiving the output of the system and go backwards towards the inputs of the system. At each step ask yourself "what is required to be able to perform this step?" Reasoning backwards has some advantages:

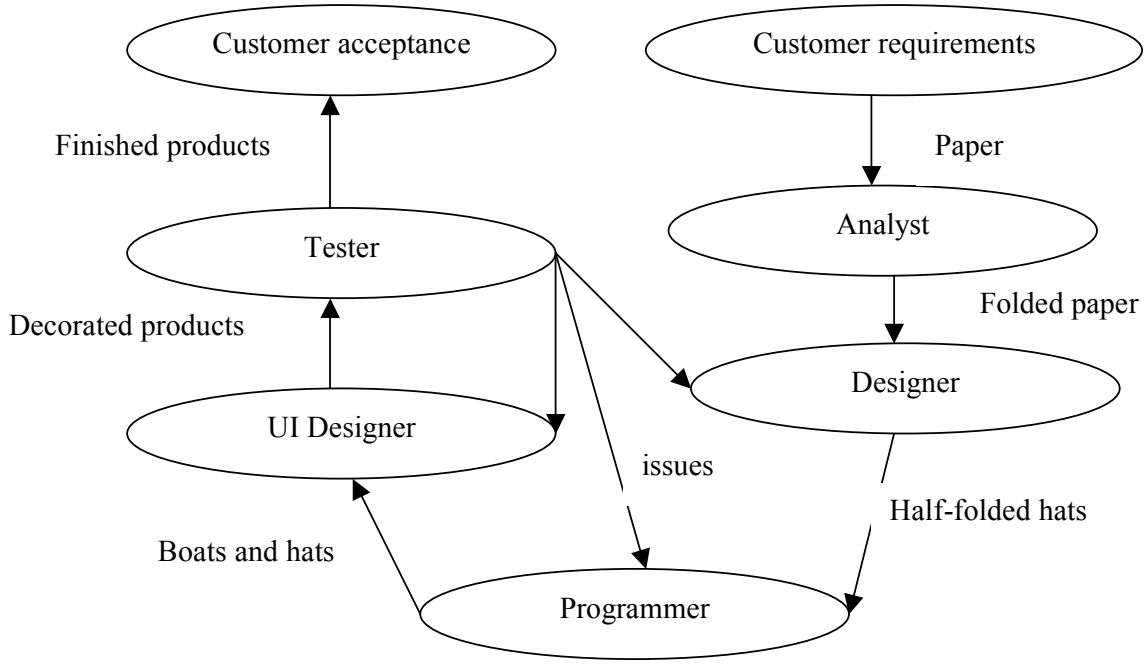
- You only consider steps that contribute to the goal

- You get into a customer-oriented "pull" mindset, where each component is seen as a "customer" of the components before it.

Draw each step on the paper sheet or on a post-it. Link the steps with dataflow arrows that describe which information and goods flow from one step to the next.

Example from the simulation:

Draw this diagram from Customer acceptance to Customer requirements.



Step 3: Find the bottleneck (5 min)

The system is now laid out on the sheet of paper. Which of its components is the bottleneck? Remember the symptoms:

- The bottleneck is very busy, all the time
- Work piles up in front of the bottleneck
- People downstream of the bottleneck are idle some of the time.

Example from the simulation:

The Programmer is the bottleneck. Mark the Programmer with a red circle. Draw a pile of work in front of them.

Step 4: Present the system (2 min per group)

Each group in turn presents their system, goal and the bottleneck they found to the other groups. The Consultants in the group present the system to demonstrate that they really understood their Customer. Each group gets a maximum of 2 minutes.

After the presentations, ask participants if they had any difficulties with any of the steps. Other participants and the coaches can provide tips to deal with the difficulties. If a group has serious difficulties, the coach can offer to help resolve them when the session restarts.

Step 5: Exploit the bottleneck (5 min)

The group brainstorms ways to exploit the bottleneck. The Consultants propose ways to exploit the bottleneck: avoid multitasking, take away non-value adding work, ensure that there is always work ready for the bottleneck… Remember: an exploit is free and involves only the bottleneck.

The Customer should react with an open mind to each proposal: don't concentrate on why the proposal can't work; explore what should happen to make the proposal work. List the proposals, but don't judge yet. Later on in the session, we will choose which proposal to implement.

Example from the simulation:

Remind participants of a few examples from the game: ensuring there is always a buffer of work in front of the bottleneck, taking bug fixing away from the Programmer, make the Programmer work on boats and hats based on real need, not a plan…

Step 6: Subordinate every decision to the bottleneck (5 min)

The group brainstorms ways to subordinate the rest of the system to the bottleneck: the bottleneck sets the pace, other resources have slack and help the bottleneck, increase quality of inputs to the bottleneck… Remember: subordination is free and involves the rest of the system, most likely the components of the system that interact directly with the bottleneck.

Again, the Customer should react with an open mind to each proposal: don't concentrate on why the proposal can't work; explore what should happen to make the proposal work. List the proposals, but don't judge yet. Later on in the session, we will choose which proposal to implement.

Example from the simulation:

Remind participants of a few examples from the game: Designer helps the Programmer, everybody works at the same pace as the Programmer, ensure high quality goes in to Programmer and no output of the Programmer is wasted.

Step 7: Elevate the bottleneck (5 min)

In the final round of improvements, the group looks for ways to elevate the bottleneck: training, better tools and environment, more resources… Because elevating actions cost time and money, the group should estimate how much each elevation will cost. Before implementing this type of change, the customer will have to demonstrate that the change has a positive business case.

Example from the simulation:

Remind participants of a few examples from the game: adding a Programmer, training, improving the workspace…

Step 8: Decide which improvement actions to take (5 min)

The Customer now has a list of proposals to exploit, subordinate to and elevate the bottleneck. The group looks at each of the categories in turn. The Customer chooses the most promising proposals and the group works out how to implement them. The Customer chooses three actions to take, preferably exploit and subordinate actions because they will be easier to implement.

Step 9: Present the actions (2 min per group)

The Customer in each team presents the 3 actions they will perform to improve their system.

Step 10: Debrief (15 min)

Advise each of the Customers to redo the exercise with the people who will be involved in the change, so that they too see the bottleneck and can come up with their own improvements. Always start the exercise with a blank page (physically and mentally): don't push solutions, do collaborative problem-solving.

Some questions to ask the participants:

- As a Customer, did you discover new insights about your system? Were you surprised by the bottleneck?
- As a Consultant, did you find the techniques this session provided helpful to understand your customer?
- As a Customer, did you feel the Consultants really understood you?
- As a Consultant, were you tempted to come up with solutions quickly? Did the focusing steps slow you down? Did you come up with surprising proposals?
- As a Customer, were you surprised by the Consultant's proposals? Did you feel tempted to react with "*That won't work!*"?

Remind participants of the fifth focusing step: "And Again!" Improvement is never done. Apply one improvement; measure the effect on throughput, investment and operating expense; go back to step 0. Set up a culture of continuous improvement by doing regular, small improvements where everybody is involved and everybody sees "the big picture".

Debriefing and closing

Ask each group to create a poster to explain the Theory of Constraints to their colleagues at work and to their loved ones at home. Each group presents their poster to the whole group in 1 minute.

Ask each individual to identify one action based on the Theory of Constraints they will commit to applying immediately upon return to work.

If you want to know more

Resources

Introductory material

- "The Goal" and "It's not Luck" by Eliyahu Goldratt introduce the Theory of Constraints in a series of business novels. Provides an easy introduction to the ideas
- "Deming and Goldratt" by Domenico Lepore and Oded Cohen presents a 10-step improvement process based on the combination of ideas of Deming and Goldratt
- "Throughput Accounting" by Thomas Corbett provides an introduction to Throughput Accounting.

More in-depth

- "Management Dynamics" by John and Pamela Caspari explains throughput accounting in more detail and provides a link with continuous improvement
- "Project Management in the fast lane" by Robert Newbold and "Critical Chain Project Management" by Lawrence Leach apply the Theory of Constraints to project management. Challenging reading, but contain lots of good reasoning and techniques
- "Thinking for a change" by Lisa Scheinkopf explains the Systems thinking tools of ToC. Interesting material, but very poorly organized and explained.

About the authors

Pascal Van Cauwenberghe

Pascal is a consultant based in Brussels, specialising in software process improvement and project management using Lean, Theory of Constraints, Agile and Systems Thinking. Over the last 15 years he has implemented systems in various domains as developer, manager and consultant.

He started the Belgian XP users group and developed the "XP Game" with Vera Peeters. He's one of the organizers of XP Days Benelux and a regular speaker at agile events.

Website: http://www.nayima.be

Portia Tung

Portia is a consultant based in London, specialising in software process improvement using principles and practices from Lean Software Development and Scrum. Portia has had a number of roles over the years, ranging from Java developer and technical team lead to development manager and consultant. Portia typically works in a multi-disciplined and technical capacity, helping organisations become more agile through collaboration and coaching.

Website: http://www.portiatung.org

Acknowledgments

Thank you to Vera Peeters, Rob Westgeest and Marc Evers for their ideas and for co-presenting the session.

Thank you to the participants in the sessions at user groups, conferences and at our customers for playing and feedback.

Session materials

Job Instruction sheets

One job instruction sheet is given to each of the players. The sheet describes the tasks that the player must perform. The sheet for the first and last player (Requirements and Production) contains a grid to record the number of papers used and the number of boat + hat pairs produced. The Job Instruction Sheets are included as Appendix 1.

Writing material

The following roles need writing material:

- Requirements: pen to record the pieces of paper used
- Analyst: pen or pencil to draw a line for the fold
- User Interface specialist: colour pens or pencils to draw flowers, portholes and anchor
- Production: pen to record the pairs boat + hat produced.

Folding Paper

Provide plenty of paper in different colours to the Requirements player. The colour of the paper is not important for the result, but makes the session more playful and allows the players to encode information. For example: "yellow paper for hats and red paper for boats."

We use A4 sheets of 80g paper, cut in two to create 21cm x 13,5cm sheets. Don't worry if *some* pieces are not cleanly cut. The Requirements player has to perform quality control and not use poorly cut sheets

Chocolates or candies

Use candy or chocolates to 'pay' and reward the team. Each player gets paid one piece of candy per round (wages). The team gets one piece of candy per accepted pair of boats and hats, to divide amongst team members.

Dice

One die per player to let players gain new skillsets (see "Step 3: Subordinate every decision to the bottleneck").

Flipchart / whiteboard

Use two flipcharts or whiteboards:

- One to record the 5 focusing steps and any important information about the theory
- One to record the results of the game.

Workshop materials

Give each group a few sheets of flipchart paper, post-its, pens and pencils.

Handout

Print out or provide an electronic copy of the handout in Appendix 2 to all participants after the session.