Indian Journal of Science and Technology, *Vol 8(29), DOI: 10.17485/ijst/2015/v8i29/85288, November 2015*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Identifying Bottlenecks in Agile Software Development using Theory of Constraints Principles

Ananthi Sheshasaayee[1]* and Hannah Vijaykumar[2]

[1]PG and Research Department of Computer Science, Quaid-e-Millath Government College for Women (Autonomous), Anna Salai, Chennai – 600002, Tamil Nadu, India
[2]Department of Computer Science, Quaid-e-Millath Government College for Women (Autonomous), Anna Salai, Chennai - 600002; Tamil Nadu, India

Abstract

Theory of Constraints is an analytical tool. A theoretical model for identifying bottlenecks in agile software development methods namely Lean software development, Extreme Programming (XP), Scrum, and Feature Driven Development (FDD) is identified and comparitative analysis is done. Identifying bottlenecks narrows down possible issues in agile software development implementations. These practices are used as surgical tools for various development models to be deployed for software development system. This paper summarizes Targeting with TOC, Eliminating the Constraint and Elevating the Constraint with Agile Principles and Practices. Theory of Constraint is concluded as the best parameter for improvement of development model.

Keywords: Agile Software Development, Bottleneck, Extreme Programming (XP), Feature Driven Development (FDD), Lean Software Development, Scrum, Theory of Constraints (TOC)

1. Introduction

Agile software development methods have different approach but the same goal[1,2]. Every development method has limitation in process and identifying such weaknesses enables organizations or developers to gain more benefit for development. Agile is not an exception and has bottlenecks which reduce its productivity. In this paper Lean, Scrum, XP and FDD software methodologies are chosen to identify the weakness in each method and to achieve an overall model. Every process has a bottleneck – a weakest link in the chain that limits throughput. Identifying and eliminating it will increase throughput what leads to more profit[1,3]. The research question which can evolve will be what are potential bottlenecks in agile software development? The main concept of TOC is to identify and exploit bottlenecks and TOC principles is used to identify possible bottlenecks in agile software development projects.

Most agile methods attempt to minimize risk by developing software in short time boxes, called iterations which include planning, requirements analysis, design, coding, and documentation. Agile methodology is significantly important in software development with regard to geographical scope and development process to achieve cost efficiency and improve customer satisfaction. The size of projects is comparatively small with high level of customer involvement and documentation.

2. Theoretical Overview

2.1 Theory of Constraints (TOC)

Goldratt developed an approach for continuous improvement called Theory of Constraints (TOC)[3,4]. TOC is applied for production, manufacturing operations and supply chain management. We apply TOC thinking

*Author for correspondence

principles to identify potential bottlenecks in agile software development. TOC is a prescriptive theory and it has developed tools to make logical decisions how to deal with them. TOC enables managers to answer three fundamental questions about the change:

- WHAT to change?
- What to change TO?
- HOW to cause the change?

These questions are system-level and not process-level questions. They are designed to focus efforts on the whole system improvement. A system is a project or a portfolio of projects in software development environment. TOC focuses on bottlenecks which allow increasing throughput of a project or a project portfolio. Exploiting identified bottlenecks will definitely affect internal activities within the project which might make them less efficient. Despite that, the throughput of the system as a whole will be increased.

2.1.1 TOC Principles

These are some of TOC principles used to identify bottlenecks in agile software development:

2.1.1.1 System as "Chains"

TOC views every system as a chain or a system of chains[5]. This is an essential way of thinking which implies that every chain has the weakest link, a bottleneck. Furthermore, at the particular point of the time there is the only one weakest link, which enables clear focus. The weakest link (bottleneck) can be found and strengthened. Working only with the weakest links will improve the system (chain) as a whole.

2.1.1.2 Cause and Effect

Every system exists in cause-effect relations. Something happens (the effect) because something else has happened (the cause)[6]. TOC provides tools and a thinking process to employ cause-effect relations to represent our complex environments. They are visually presented as trees. In order to achieve higher branches in a tree, all lower ones must be implemented.

2.1.1.3 Undesirable Effects and Core Problems

Almost everything found in a system as problems are actually undesirable effects. Solving undesirable effects gives false security feeling that a problem is solved. Nevertheless, the existing problem has a tendency to appear again as a core problem still exists in a system. Only after the core problem is solved, the undesirable effect, that was a bottleneck in the first place is actually solved and prevented from returning. Identifying core problem means identifying the bottlenecks.

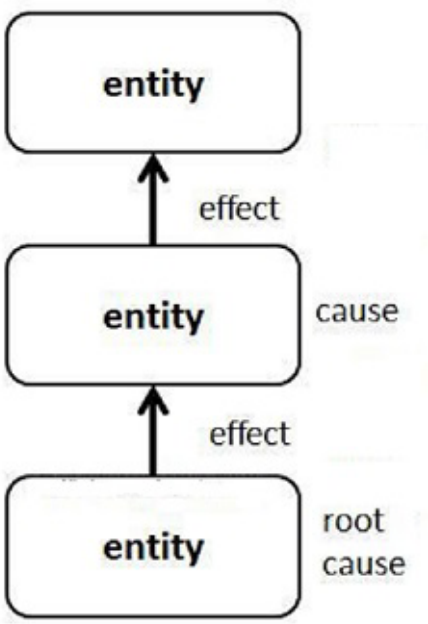

Figure 1. Cause – Effect structure.

2.1.1.4 Physical vs. Policy Bottlenecks

Physical bottlenecks are relatively easy to find and break but most real bottlenecks that exist in systems are policy bottlenecks. Physical bottlenecks are just a result of policies and rules in organization. All software documentation policies in company have to be reviewed and changed accordingly.

2.1.2 The Five Focusing Steps

These steps are followed in order to break the identified bottlenecks and to continually improve the agile software development projects. The five focusing steps are:

2.1.2.1 Identify the System Bottleneck

Find the weakest link in the system of chains. There is only one weakest link at a given point of time.

2.1.2.2 Exploit the Bottleneck

When a bottleneck is found it is essential to assure that it works 100% and all activities which do not directly add value to the tasks of a bottleneck has to be eliminated.

2.1.2.3 *Subordinate Everything else to the Above Decision*

After performing step 2 the rest of the system has to be adjusted to enable a bottleneck to operate with maximum effectiveness.

2.1.2.4 *Elevate the System's Bottleneck*

This step is reached in case steps 2 and 3 did not break the bottleneck. Elevating the bottleneck means doing whatever it takes to break it. This step should be executed only after doing everything that is possible in steps 2 and 3.

- Go back to Step 1, but do not allow inertia to become a system bottleneck. There is always the weakest link in a chain. If a bottleneck is broken in step 3 or 4 it is a must to come back to step 1 and start looking for a new bottleneck. This is the process of continuous improvement which never ends.

There are four main motivation factors to use TOC and its principles as a research tool. First, TOC principles enable to view agile software development as system of chains. Second, they allow modeling agile software development principles and practices into trees with cause-effect relations. Third, TOC principles enable to identify main bottlenecks and finally TOC allows focus on core problems and to channel improvement efforts for maximum immediate effect. This means that the identified possible bottlenecks can be immediately reviewed and exploited in an organization to achieve fast results. This also creates a process of continuous improvement.

2.2 Agile Principles

Agile software development emerged as an alternative to document driven, rigorous software development processes. Software developers realized that processes which require many documents, artifacts and procedures to follow is too slow to fulfill customer needs[7]. Hence the focus had to switch from fulfilling well predefined project requirements to delivering up to date value to the customer.

2.2.1 *Manifesto for Agile Software Development*

A common ground for Agile Software Development was defined in 2001 when 17 experienced and recognized software development "gurus", inventors and practitioners of different agile software development methods gathered together and signed The Manifesto for Agile Software Development. This manifesto declares the main values of agile software development: "We are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

2.2.2 *Principles Behind the Agile Manifesto*

Manifesto for Agile Software Development is followed by 12 principles which are used as a base for identifying possible bottlenecks in different agile software development methods[8].

- Satisfy the customer.
- Welcome changing requirements.
- Deliver working software frequently.
- Motivate individuals.
- Interact frequently with stakeholders.
- Communicate face to face.
- Measure by working software.
- Maintain constant pace.
- Sustain technical excellence and good design.
- Keep it simple.
- Empower self-organizing teams.
- Reflect and adjust continuously.

3. Agile Software Development Methods

There is a number of agile software development methods that follow the values and principles described above. The main are: Lean software development, Extreme Programming (XP), Scrum, and Feature Driven Development (FDD).

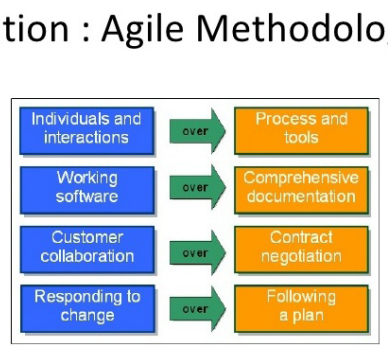

Figure 2. Agile Principles.

3.1 Lean Software Development

Lean thinking focuses on giving customers what they want, when and where they want it, without a wasted motion or wasted minute.

Lean Software development process is composed of four phases:

- Preparation.
- Planning.
- Implementation.
- Assessment.

At the beginning an initial backlog of prioritized desirable features is assembled. This is the preparation phase. Backlog items usually features in terms of business goals since the Lean approach is to delay detailed analysis until the last responsible moment. Planning meeting is held at the beginning of each iteration. The whole team makes estimations how long the top priority stories from backlog will take to develop, test, document and deploy. As per these estimations and team capacity they pick the amount of stories they will be able to implement during the iteration. Team members decide and commit to iteration goal. During implementation phase a team develops, tests, documents and prepares for deployment of the feature set they picked. Daily 10-15 minute team meetings are held to discuss what each team member has accomplished since the last meeting, what they will be doing till the next meeting, what problems they have, and where they need help. A story is not considered done until the team updates all associated artifacts. A review meeting is held at the end of each iteration. The goal of the meeting is to show for the customer how much value was added to the product during the iteration. Feedback from customer is collected to make changes if needed[9]. After this iteration assessment, planning meeting starts for the next iteration.

Lean Software development has 7 main principles

As a group, these principles provide guidance on how to deliver software faster, better and for less cost.

- Eliminate waste – remove everything that does not create a clear value for a customer (product).
- Build quality in – focus on eliminating defects as soon as they are detected; avoid creating defects in the first place.
- Create knowledge – encourage systematic learning throughout the development cycle and make sure that knowledge is shared.
- Defer commitment – schedule irreversible decisions for the last responsible moment, that is, the last chance to make the decision before it is too late.
- Deliver as fast as possible – deliver software so fast that your customer would not have time to change their minds.
- Empower the team – develop an organization where each person has an authority to prioritize, take responsibility and come up with solutions instead of having someone telling what to do and how to do it.
- Optimize the whole – optimize the whole value stream from the time it receives an order to address a customer need until software is deployed and the need is addressed.

3.2 Extreme Programming (XP)

Extreme Programming (XP) is an agile development method that values simplicity, feedback, community, and courage. XP development process consists of three main iterative phases:

- Planning.
- Development.
- Acceptance.

At the beginning of a project, Planning Game is held where the project is divided into iterations of 1 to 3 weeks. Story Cards that represent features are created and the project releases dates are set. A task list is created and team members choose the tasks they want to work next. Development phase starts with the high level design sketch on a whiteboard. Programming is held in pairs where both team members have the same responsibility for the code. All code is continuously integrated and tested on a separate machine. In the acceptance phase

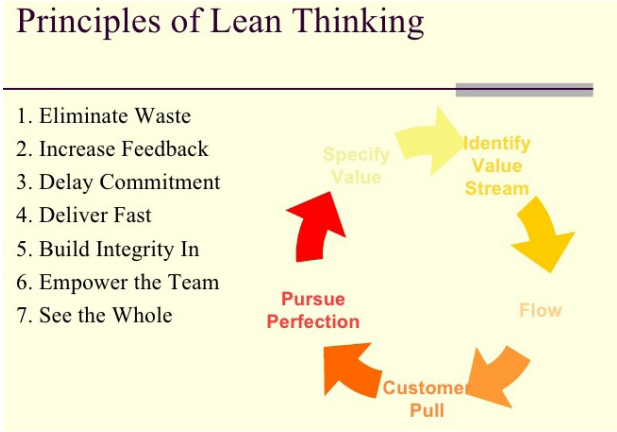

Figure 3. Lean Principles.

all code is tested with automated acceptance test that is defined by a customer. A review meeting is held to get the feedback[10]. XP software development recommends these principles:

- Planning game – a planning session where story cards are defined and prioritized together with a customer.
- Test-first development – a development culture where first a unit test is created and afterwards the code is written.
- Simple design – a design that has a main set of classes and methods and is created only when it is needed. No generalized components are created if not needed.
- Stand up meeting – a short 15-20 minutes daily meeting where each team member answers 3 main questions:
  What is done so far?
  What is planned to do until next meeting?
  What are the obstacles to achieve iteration goals?
- On site customer – a working process where one or more customers are in the same room as a development team full time.
- Continuous integration – an integration activity where all code is continuously integrated in a common environment where the unit tests are run continuously.
- Short releases – an evolutionary delivery to increase suitability for business needs.
- Acceptance test – an automated acceptance test that is run with pass/fail result which is defined by a customer.
- Collective ownership – a development culture where any pair of programmers can improve any code creating an environment that no-one is blamed for mistakes in code.
- Common room – a working environment where the whole team is working as close as possible preferably in one room.
- Frequent refactoring – an effort to simplify the code to make it cleaner without changing its functionality.
- Coding standards – a coding style accepted by a company to ease coding and refactoring processes.
- 40 hours week – a working culture where work is limited to increase creativity and avoid overtime.
- System metaphors – memorable metaphors to enable better understanding about system in the design sketches.
- Pair programming – a development culture where all code is produced by two programmers at one computer.

3.3 Scrum

Scrum is an agile software development method that provides a management framework:

There are four basic phases in this process:

- Planning.
- Staging.
- Development.
- Release.

The goal of the planning phase is to set the vision and expectations of a project and assign funding. This is done in pregame planning. Moreover, the project is divided into iterations called sprints that are 30 calendar days. In the staging phase requirements should be identified and priorities assigned for the iteration. This phase begins with sprint plan where the plan for iteration is created. External stakeholders are involved to prioritize the tasks in sprint. No more additional tasks can be added to sprint after the plan is created. The development phase involves a system implementation in 30 days iterations and prepares it for a release. During this phase the work in sprint is divided into daily blocks that lead to daily builds.

The development begins with high level design sketches. Every day a 15 minutes stand up meeting is held to update on the sprint status. During the meeting team members choose the tasks they will be working next. During the last phase the system should be deployed. After each sprint the release meeting is held where a system to external stakeholders is presented to get feedback. After that future directions are set.

Scrum development process recommends these principles

- Scrum meeting – a short 15-20 minutes daily meeting where each team member answers 3 main questions:
  What is done so far?
  What is planned to do until next meeting?
  What are the obstacles to achieve iteration goals?
- Sprint – 30-days iteration.
- Pre-game planning – a planning activity where the product backlog is created with list of features, use cases and defects as well as product owner is assigned to ease future communication.
- Sprint planning – a planning activity that consists of two meetings: first, stakeholders refine and prioritize product backlog, second, team and product owners plan how to achieve iteration results and create task lists.

- Common room – a working environment where the whole team is working as close as possible preferably in one room.
- Daily built – at least a single integration with a regression testing of the code in the system throughout a day.
- Blocks gone in one day – tasks that are finished in one day.
- Scrum master firewall – Scrum master/manager activity to assure that work in team is happening and no undesired activities exist within the team.
- Lock priorities within sprint – priorities chosen at the beginning of sprint. No extra work that could be added to iteration is tolerated to maintain team focus on the goal. In case extra work is added some work should be removed.
- Sprint review – a meeting where a review for sprint is executed and demo of a product is presented at the end of sprint. Feedback and future directions are set during this meeting.
- Decision is one hour – decision making process that does not take longer than one hour. No decision is worse than a bad decision and a bad decision can be reversed.
- High level design – the sketch of design only to get basic understanding about the system.
- Self-directed and self-organizing teams – the culture where the team has authority and resources to choose the best way to achieve sprint goals, to prioritize work, and to solve its own problems.
- Team of 7 – the team that consist of no more than seven people to assure efficiency and smooth communication[11].

3.4 Feature Driven Development (FDD)

Feature Driven Development (FDD) is an agile software development method that values up front modeling and has "right first time" approach. It is a minimally described five steps process. First three steps are executed once in a project and called "startup phase" while steps 4 and 5 are iterated for feature sets and called "construction phase".

- Develop an overall model.
- Build a features list.
- Plan by feature.
- Design by feature.
- Build by feature.

First, the overall model is developed. The model is very brief and it contains only main classes and their connections. In larger projects the domain teams are formed and domain models are created by different teams. They are merged into overall model daily or every second day.

In the second step the complete and categorized feature list is built. To compile this list the domain is decomposed into Subject Areas. Then Subject Areas are decomposed into Business Activities and the features within each Business Activity. The size of a feature usually is from one to ten days work. The goal of plan by feature stage is to produce a development plan. Planning Team consisting of Project Manager, Development Manager and the Chief Programmers is created. The team plans an order that features have to be implemented according to feature dependencies, complexity and the load of development team. Chief Programmers are assigned to Business Activities and developers are assigned to own the classes. Chief Programmer selects a feature set from his entire features list called Chief Programmer Work Package. Then he forms Feature Team by identifying the owners of the classes which will be involved in the development of a selected feature set. This team creates needed design for Chief Programmer Work Package which is refined against overall model created in the first step. Finally, Feature Team implements Chief Programmer Work Package by following these steps: implement classes and methods, inspect the code, run unit tests, and promote to the build. After the build succeeds, new iteration from step 4 starts with a new Chief Programmer Work Package and a new Feature Team[12]. As every agile software development method, FDD has main principles it follows:

- Domain Object Model – a process of creating the framework of problem domain within which features will be added.
- Development by Feature – a process where development is driven and tracked by decomposed list of small, client valued functions.
- Individual Class (Code) Ownership – a process where the consistency, performance, and conceptual integrity of each class is the responsibility of an assigned single person.
- Feature Teams – a process encouraging doing design activities in small, dynamically formed teams as well as encouraging evaluating multiple design options before one is chosen.
- Inspections – a process of defect-detection technique providing opportunities to propagate good practice, conventions, and development culture.

- Regular Builds – a process ensuring that there is always a demonstrable system available. It also helps to solve all synchronization issues as early in the process as possible.
- Configuration Management – a process ensuring an easy way to identify/revert/change any version of the completed source code files and other artifacts of the project.
- Reporting/Visibility of Results – a process of frequent and accurate progress reporting at all levels, inside and outside the project, based on completed work.

4. Implementation Methodology

Every process has a limitation which prevents organizations to increase its product manufacturing, the main goal of any organization. Many case study researches have shown that among all development methods, agile is used more in organizations. This paper helps us to study of creating a model to identify weakest parameter in agile methodologies. The research can be divided into phases[13], why specific methods are chosen, and how they contribute to find the answer to the main research question:

4.1 What are Potential Bottlenecks in Agile Software Development?

To answer this question, the research is divided into three phases with a question in each phase.

- Question 1: What high level bottlenecks might exist in agile software development methods?
- Question 2: What bottlenecks might exist in Lean software development?
- Question 3: What bottlenecks might exist in agile software development implementation in a case study under research?

Phase 1: Identify possible high level bottlenecks of agile software development methods.

The goal of phase 1 is to identify high level bottlenecks of agile software development methods. In this we refer to high level bottlenecks as missing or not directly addressed principles and practices of agile software development methods. To accomplish this goal, first, we have to get a clear understanding of the agile software development principles and their application in the specific agile software development method. We have to identify the main principles and understand the differences between various agile software development methods.

Having this knowledge allows us to define the question "What high level bottlenecks might exist in agile software development methods?" To answer the above question we make an assumption that each successful agile software development method has to address all general agile principles agreed by authors of Manifesto for Agile Software Development. We expect that different agile software development methods focus on different agile principles. We use software to code all principles and practices of analyzed agile software development methods. We use general agile principles agreed by authors of Manifesto for Agile Software Development as codes. Afterwards analyzing the TOC principles, model is generated for study. Each principle and practice of each analyzed agile software development method is connected to general agile principle in it. We expect to find the gaps where specific agile development method does not directly address specific general agile principle. These gaps present possible high level bottlenecks.

Phase 2: Identify possible bottlenecks in Lean software development. In phase 1 we identify possible high level bottlenecks for different agile software development methods. In phase 2 we choose one agile software development method and identify possible bottlenecks for it. Furthermore, we define actions for each identified possible bottleneck. Defined actions should help us to measure if a possible bottleneck is a real bottleneck in a specific agile software development method implementation. This approach is mainly following Lean software development principles. Therefore, Lean software development is chosen from analyzed agile software development methods for this phase. Having the goal of phase 2 and agile software development method, question 2 is defined as "What bottlenecks might exist in Lean software development?" To be able to achieve success in a Lean software development project, all principles must be fulfilled. To fulfill each principle, a set of practices must be executed. Therefore, first our task is to identify practices needed to implement Lean principles. TOC principles described in paragraph are used to model the system and make visual presentation of the results. Therefore, the output of phase 2, the Lean software development tree with possible bottlenecks is called a theoretical model of possible bottlenecks in Lean software development. After we have the theoretical model of possible bottlenecks in Lean software development

created, we define actions for each possible bottleneck. These actions help to identify if possible bottleneck is a real bottleneck in a specific Lean software development implementation. These actions will be the guidelines for the questions in phase 3.

Phase 3: Identify possible bottlenecks in agile software development implementation in a case study research[14,15]. In phase 2 we develop the theoretical model of possible bottlenecks in Lean software development. We also define actions for each possible bottleneck. The goal of phase 3 is to apply the theoretical model developed in phase 2 for actual implementation of agile software development method. The question 3 for this phase is "What bottlenecks might exist in agile software development implementation in a research case study?" To answer this question the method of questionnaire is used. This method allows us to focus interviews on bottlenecks as well as to keep them open. To prepare for interviews we pre-select 7 most probable bottlenecks for agile software development implementation in a case study from all possible bottlenecks list identified in theoretical model in phase 2. We base our selection on the rule to have one bottleneck connected to each principle and our current knowledge about situation in a case study. After that, we prepare a set of open questions. These questions allow interviewees to discuss and decide whether possible bottlenecks exist in their environment. At the end of interviews, we ask interviewees to prioritize analyzed possible bottlenecks according to their influence on the whole project performance. The prioritized list of possible bottlenecks in agile software development implementation is a case study which is the expected output of the research. A sample questionnaire is listed in Appendix A.

5. Result Analysis

A comparison between principles of the analyzed agile development methods and the general agile software development principles is carried out. It enables us to find the possible high level bottlenecks in each analyzed software development method. Weak links might exist in agile software development. Creators of different software methods had a meeting and agreed on agile Manifesto which is considered as core values or principles of agile software development. All the principles of agile development are accepted and should be addressed in order to have successful agile software development method. The comparison of principles and practices of each method which were described in previous sections can be identified. Through this comparison it can be understood that there are gaps when there is no connection between principle or practice of agile method and general agile principles. Based on this comparison and identified gaps we can also find high level bottlenecks in each agile development methods.

5.1 Probable Weak Links of Lean Software Development

Lean software development method does not have principle to address lack of individual motivation. "Empower the team" may proportionally cover the lack of motivation system. Nevertheless leaders should not ignore this possible bottleneck. The way of interacting with stakeholders is not defined in Lean. This is done as a group which means a team decides how to interact with stakeholders. However, "Create knowledge" principle in Lean needs fast and continuous feedback for continuous learning. Therefore close and frequent communication with stakeholders should be carried out by teams. Even though poor communication process itself can be considered as a potential bottleneck. Against "face- to- face communication" principle in general agile, Lean just has daily short meetings each of which is 10-15 minutes. It can also be considered as a kind of good face-to-face communication. If attention is not given to this principle, selecting time consuming methods will increase the impact of lack of face-to-face communication. The main objective of agile development is simplicity. Thus lack of simplicity might be the most difficult bottleneck to break. The reason is Lean focuses on improving different process parallel and it has managerial view rather than technical view. Therefore all team members should understand the process and goals of Lean and follow them responsively.

5.2 Probable Weak Links of Extreme Programming (XP)

General agile development method focuses on frequent interaction with stakeholders, means customers, product manager, sponsors, and others involved but XP just focuses on interaction with customers. This definitely is a possible bottleneck for XP method. In addition, XP does not have any principal regard to how team should improve

the process. This bottleneck is hard to break because XP concentrates on technical development activities.

5.3 Probable Weak Links of Scrum

In Scrum there is no "ON site customer" principle because it is not required, customer just need to prioritize the Sprint backlog. Nevertheless there is no communication with other stakeholders in Scrum software development. Accordingly it is absolutely high level bottleneck for Scrum. In Scrum measures exist to show that tasks are already finished but no measure exist to know if it is accepted by customer. Therefore lack of project progress measurement can be a possible bottleneck for Scrum.

5.4 Probable Weak Links of Feature Driven Development (FDD)

FDD principles are not about customer satisfaction against general agile principle. FDD focuses more on implementing requirements and success means accomplished ones not those accepted by customers. This is a possible bottleneck. Moreover team motivation is an important principle for project success. In FDD for each iteration, different feature teams are formed therefore people have to switch between these groups frequently. In such an environment to motivate a team member can be a big issue. As a result developers face lack of team member motivation in FDD. There is mismatch between progress measurement in general agile software development and FDD progress measurement. FDD measures project progress by implemented features and not accepted features by customers. There is no review and changes are made due to customer requirements. "Individual class ownership" principle in FDD decreases face-to-face communication because each of the team members is responsible for his class consequently and has no need to communicate. Therefore lack of progress measurement by working software and lack of face-to-face communication are the possible bottlenecks for FDD. "Maintain constant pace" principle in general agile development method provide sustainable development and it is responsible of all stakeholders. However this point is not considered in FDD and there is no way to know how the knowledge should be shared. Implementing process continuously helps developers to decrease the impact of lack of continuity. FDD is designed for large project and thus it has complex process while agile development extremely focuses on simplicity.

6. Conclusions

Agile software development enables organizations to respond to the changing market so fast and is used successfully by organizations. The knowledge of people involved in agile methodologies has been increasing. Despite all difference between principles of the analyzed agile development methods and the general agile software development there are common values such as user involvement which is vital to success, to build and release software in small increments, attention to quality and excellence and conformance to standard. In this paper principle of Theory of Constraint (TOC) is used to find the bottlenecks. All identified bottlenecks in different agile software development methods should be noticed during implementing the method. To find these bottlenecks first the principle of general agile method and then principles of four selected agile methodologies were discussed. Second, bottlenecks of general agile development method are identified by comparing the general agile principles and principles of agile software development methodologies. Gaps or mismatch between these two is considered as possible bottlenecks which are identified by research case study in an organization adopting agile principals and practices.

7. References

1. Goldratt EM, Cox J. The Goal: A Process of Continuous Improvement. 2nd ed. Great Barrington, MA, USA: The North River Press; 1994.
2. Goldratt EM, Schragenheim E, Ptak C. Necessary but not sufficient. Great Barrington, MA, USA: The North River Press; 2000.
3. Anderson DJ. Agile management for software engineering: Applying the theory of constraints for business results. Berlin, Germany: Springer; 2003.
4. Bohem B. Get ready for agile methods with care. IEEE Computer Society. 2002; 35(1):64–9.
5. Cohn M, Ford D. Introducing an agile process to an organization. IEEE Computer Society. 2003; 36(6): 74–8.
6. Sommerville I. Software Engineering. 9th ed. England: Addison-Wesley; 2007.
7. Abrahamsson P, Salo O, Ronkainen J. Agile software development methods: Review and analysis. VTT, Finland: VTT Publications; 2002. p. 478.
8. Cockburn, Alistair. Agile Software Development. Boston, USA: Addison Wesley; 2002.
9. Poppendieck M, Poppendieck T. Lean Software Development: An agile toolkit. Boston, USA: Addison Wesley Professional; 2003.

10. Beck K. Extreme Programming explained: Embrace change. Boston, USA: Addison-Wesley; 1999.
11. Cohn, Mike. Succeeding with Agile: Software development using Scrum. Boston, USA: Addison-Wesley; 2010.
12. Palmer SR, Felsing JM. A practical guide to feature-driven development. Upper Saddle River, USA: Prentice Hall; 2002.
13. Newbold RC. Project Management in the fast lane: applying the theory of constraints. St. Lucie Press; 1998.
14. Cao DB. An empirical investigation of critical success factors in agile development projects [Phd thesis]. Minneapolis, USA: Capella University; 2006.
15. Daniel T, France R. Assumptions underlying agile software development processes. Journal of Software and Systems Modeling. 2003; 17: 19–25.

Appendix A

Questionnaire Sample - Case study:

1. Do you experience that artefacts have waiting periods? Where usually is the longest artefact inactivity moment? (Waiting for resources, waiting for approvals, and other waiting periods).
2. How do you perform synchronization? Do you encounter any problems due to continuous synchronization? If so, could you name the top problems?
3. Does your team have regular feedback sessions? If yes, how often? Which problems are discussed most often? Is a customer involved into these feedback sessions?
4. Is there a practice in a company to create several solutions (or one adaptable) for the complex problem? If yes, when and how? If no, why not? When is the final decision made?
5. Do you have a backlog of features prepared for iterations? How much work (in person hours) is there in the list? How much time do you spend on managing the backlog of features?
6. Have you identified the competences of each team member in your teams? What processes do exist to share their knowledge with others?
7. How do you decide which processes to improve? Are there any specific measurements that influence the decision? Do those measurements focus on local optimization? Are they evaluated against the impact to the whole system (project)? Could you exemplify both?