Bottlenecks Identification in Software Development Process: A Quali-Quantitative Approach

Conference Paper · October 2017

DOI: 10.1145/3160504.3160513

4 authors:

Sildenir Alves Ribeiro
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
11 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Eber Schmitz
Federal University of Rio de Janeiro
65 PUBLICATIONS **175** CITATIONS
SEE PROFILE

Antonio Alencar
Federal University of Rio de Janeiro
54 PUBLICATIONS **270** CITATIONS
SEE PROFILE

Mônica Ferreira da Silva
Federal University of Rio de Janeiro
33 PUBLICATIONS **42** CITATIONS
SEE PROFILE

Bottlenecks Identification in Software Development Process: A Quali-Quantitative Approach

Sildenir A. Ribeiro[1,2] Eber A. Schmitz[2] Antônio Juarez S. M. de Alencar[2] Mônica F. da Silva[2]

[1]Coordination of Industrial Automation
Federal Center of Technological Education of Rio de Janeiro (CEFET/RJ)
Rio de Janeiro, Brazil
sildenir.ribeiro@cefet-rj.br

[2]Tércio Pacitti Institute of Research and Computer Applications
Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
{eber, ajuarez, monica}@nce.ufrj.br

ABSTRACT

This paper presents the quali-quantitative results of a study on the identification of bottlenecks in the software development process. The research was developed in an environment learning of Software Engineering and had the collaboration of students of the Department of Computer Science (DCC) of the Federal University of Rio de Janeiro (UFRJ). The main objective of the research is to verify the existence of productive bottlenecks in Software Development Process (SDP), typify them and if possible to promote treatments to solve the impacts in the process and in the software product. For this, three experimental rounds were carried out and three different domains were applied. Each round had deferential teams working in the same domain. The experiments also involved the Unified Process (UP) to guiding the SDP and the Theory of Constraints (TOC) to identify and treat the restrictive elements found in the productive process. As results, the work presents a set of qualitative constraints subdivided into two groups: (1) Behavioral Constraints (BC), and (2) Technical Constraints (TC). The identification of these qualitative constraints allowed quantitative bottlenecks to be detected. The research also showed that bottlenecks are associated with certain tasks performed in the software development environment within a given domain. In general, these tasks are consuming resources, such as effort, time and human resources. This fact leads to the software product to a low quality due to the lack of completeness and correctness of the artifacts delivered to the end of the productive process.

Author Keywords

Software development process, Theory of constraints, Bottlenecks identification, Quali-Quantitative Analysis.

ACM Classification Keyword

D. Software: D.2 Software Engineering: D.2.8 Metrics: *Software Science*; D.2.9 Management (K.6.3, K.6.4): *Productivity*. K.6.3 Software Management (D.2.9): *Software development*, *Software process*; H. Information Systems (H.1): H.1.1 Systems and Information Theory (E.4): *Human factors.*

INTRODUCTION

This work began from a research opportunity identified in a review of the literature on the application of the Theory of Constraints (TOC) in the Software Development Process (SDP), presented by [21]. The secondary study of [21], presents several gaps in the application of TOC in the SDP. One of them is the identification and treatment of productive bottlenecks in the scope of software construction.

An important factor that encouraged this research was the quali-quantitative approach imposed by TOC during the investigation process to detect and treat bottlenecks in SDP.

Why TOC?

The Theory of Constraints (TOC) is a methodology widely applied in process improvement of manufacturing processes [11, 20, 23, 24]. According to [6, 7, 17], the TOC can be described as a philosophy of continuous improvement, which has evolved and expanded its methodological basis over time. TOC has been the subject of a substantial amount of research involving its application to the manufacturing process. TOC management philosophy incorporates a practical aspect to decision-making within the production environment, based on the principle that any limiting factor of a system output points to a system constraint.

TOC states that every system has at least one constraint [6, 7, 9]. A constraint is any value that can prevent a system from achieving its goal [9]. According to [6, 7], a constraint can be external (not physical) or internal (physical). The external constraints are usually associated with circumstantial problems, such as: (1) market demand: production over market capacity or production below market capacity; and (2) a corporate procedure: in this case, a decision or a procedure that limits the gain. The internal constraints are usually physical and are associated with the resources. According to [6, 7] the external constraints are of three types: (1) equipment: this is related

to the usability and productive capacity of the equipment; (2) people: involves the lack of qualified people, the productive capacity of the people and behavioral aspects; and (3) policy: the adoption of written policies, such as: laws, standards and regulations, can be a limiting factor for the productive system.

These concepts accredit TOC as a powerful tool to identify and treat restrictive elements, especially the qualitative problems of the Software Development Process (SDP), mainly internal constraints (type 2), that involves peoples by: (1) lack of technical qualification; (2) productive capacity; and (3) behavioral aspects.

Unified Process (UP) as SDP

The software development process, as well as any manufactured product, requires the same or even more insightful demands during its construction cycle. As any manufacturing process, SDP is influenced by internal and external variables. Furthermore, the characteristics of the development teams and ethnological issues have direct influence on the development process, making the development environment complex and highly dynamic.

TOC have as fundamental principle the idea that every process can be continuously improved [6, 7, 11]. For this, TOC requires that the process is mature, configurable and applicable. We promote some modifications in the UP to align with the interests and objectives of this research and we denominated of UP-Like.

The research methodology criteria used and the working methods are supported by: (1) studies in the literature about the use of UP in software development projects and (2) TOC concepts for bottleneck identification such as [6, 19, 21].

Human Factors

Differently of the processes performed in the manufacturing industry, where most of the production process is executed by machines, the SPD require of the people to execute it. This implies that human factors technical and behavioral must be considered, once they have a direct influence on software production.

RELATED WORK

Human and ethnographic factors are discussed in software engineering for decades. The qualitative aspects around software development are a broad front of research and many papers have addressed the theme in an attempt to highlight phenomena that can reduce impacts on the productive process of software.

In this work, a brief literature review was conducted to search for primary studies that have investigated quali-quantitatively the SPD. The following topics present some of these works.

- Clarke & O'Connor [4]: carried out a study that made a substantial body of related research into an initial reference framework of the situational factors affecting the software development process. The research includes: (1) the nature of the applications under development; (2) team size; (3) volatility requirements; and (4) personnel experience.
- Rainer & Hall [18]: explored a set of 26 factors that potentially affect software process improvement (SPI). The study made a qualitative and quantitative analysis of case studies, comparing results collected in a case study with the results of a previously conducted survey study. The work is outside the search period (2010-2017), but was selected due the strong correlation with this study.
- Ribeiro et al. [21]: conducted a research involving students of software engineering according to [3] where behavioral aspects of software development teams were observed. The work evidenced a new phenomenon that was called of "deadline syndrome".
- Lee & Xia [12]: performed an analytic study on the agile software development process by empirically examining the relationships among two dimensions (1) software development agility: software team response extensiveness, software team response efficiency, team autonomy, team diversity; (2) aspects of software development performance: on-time completion, on-budget completion, and software functionality.

Other works also contributed to form knowledge about this subject, as: Ribeiro et al.[22] that presented a short paper to report quasi-experiment in academic environment to detect bottlenecks SPD; Adolph et al. [1] with an empirical research that grounded a theory to study the experience of software development; Buse & Zimmermann [2] with the study on Information needs for software development analytics; Gousios et al. [8] that performed An exploratory study of the pull-based[1] software development model; McLeod & MacDonell [13], that presented a survey on factors that affect software systems development project outcomes and Nilsson et al. [15] that applied a methodological description to Assessing the effects of introducing a new SDP in the software industry.

EXPERIMETAL DESIGN

The experiment was conducted *in vivo* and in a controlled environment, as recommended by [25]. We create a specific methodology for execution and conduction of the experiment. and a model of execution of the process according to the UP-Like phases and according to the experiment template.

Our purpose is to present a model of the experiment, as defined in [10, 14, 25], following the scientific method, described in [5]. By specific needs of this work, we adjust the scientific method proposed by [5] according to the five steps of the Theory of Constraints (TOC) and the phases of the Unified Process (UP-Like). Thereby, was possible to

[1] *Pull-based development is an emerging paradigm for distributed software development (Gousios et al. [8].*

establish a cyclical model that can cover the entire software development process. Figure 1 show the scientific method.

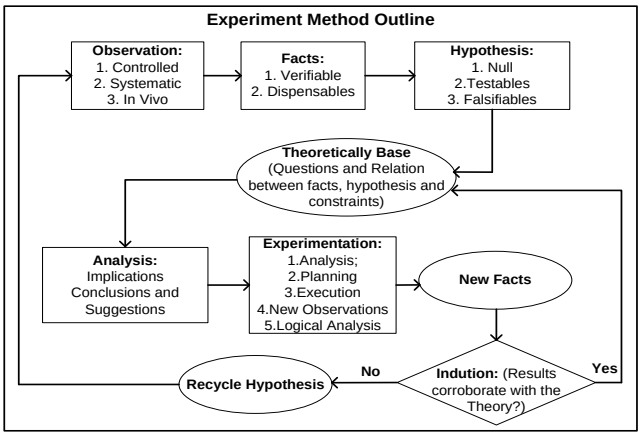

Figure 1. Scientific method applied *(adapted of [5])*

Methodology and Process

The methodological criteria used and the working methods (process) are supported by: (1) studies in the literature about the use of UP in software development projects and (2) TOC concepts for bottleneck identification such as [6, 7, 16, 21]. Figure 2 illustrates the experimental flow a sequentially oriented by nine steps, following strictly the TOC principles and the structure of Unified Process (UP).

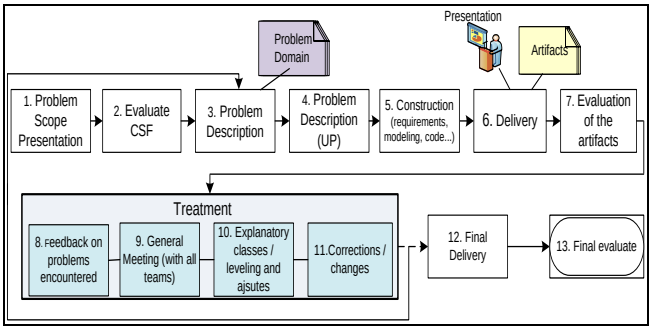

Figure 2. Experiment execution flow

Research Organization: Criteria Adopted

The research was organized with the following guidelines:

1. Random division of the class in working groups / development teams;

2. Initial team assessment using questionnaires to identify Critical Success Factors (CSF). Repeated later at different times;

3. Presentation to the teams of the product specification for the software to be developed;

4. Definition of resources to be used by all teams: hardware, software and laboratories with IT infrastructure;

5. Definition of the methodology applied in the experiment and the experimental process.

6. Creation of the experiment logbook to record the daily observations on the execution of the experiment;

7. Presentation of the project schedule: setting out delivery dates for the artifacts produced in each phase of the experiment;

8. Definition of the artifacts to be delivered: (1) Documentation; (2) Database; (3) Logical Architectural; (4) Codes (includes screens); And (5) tests. The artifacts were grouped by type of tasks and oriented by the phases of the UP.

9. Team evaluation: during the development stages according to the delivery of the artifacts.

10. Elaboration of criteria to align the artifacts delivered by each team in each stage.

11. Final evaluation: realized after the final delivery of the product, that consists of software product installed tested and approved.

12. Homologation of the product in a production environment.

Format and Minimum Requirements of the Artifacts

In order to format the productive results to be delivered by development teams, the following criteria were adopted:

- Deliveries must be made in accordance with the phases of the UP-Like;
- Deliveries must be made on the dates foreseen in the schedule / schedule of the experiment;
- The minimum requirements of deliverables comprise of a set of tasks that must be performed for each artifact constructed;
- Deliveries must meet pre-defined minimum requirements and be developed following the UP-like phases;
- The defined artifacts are: Architecture, Database, Coding, Documentation and Testing;
- The set of tasks to be performed is fixed and previously defined;
- Tasks can be processed and reprocessed by development teams until concluded (complete and correct);
- Tasks can be processed and reprocessed by different team members in different phases of UP-like;
- Tasks are scheduled according to the type (artifacts) and according to UP-Like phases;
- At each stage of the UP-Like, new tasks are inserted for the completeness of the artifacts to be delivered.

Measurement Criteria

To create a standardized assessment that could be applied commonly to each artifact produced by each development team, the following criteria were implemented.

1. Total or complete delivery: All tasks were produced and delivered. In this case, weight 1 is assigned.
2. Partial delivery: the quantity delivered is between (> 50% <100%) of the tasks to be carried out in the phase. In this case, weight 0.5 is assigned.
3. Undelivered artifact: the tasks were not produced or a quantity less than 50% was delivered. In this case the assigned weight is 0.

Software Product Acceptance Criteria

The software produced in the three experiments was evaluated through: (1) resources used; (2) environment and team of development; and (3) process and the generated software product**.**

ISO/IEC 25010:2011 and ISO/IEC 25020-24:2010 recommend a "ranking" system with scores that meet the following criteria: Satisfactory {3-Execellent, 2-Good, 1-Regular} and Unsatisfactory {0-Bad, 0-Undeliverable / Nonexistent}.

Acceptance of the artifacts at each stage of UP-Like (partial delivered) and of the final product was followed by the guidance of ISO/IEC 8402:2006, through the following parameters: functionality, reliability and usability.

For each delivery a weighted average is calculated according to the weights given in the "Measurement Criteria" session. The artifact with mean *($\mu \geq 75\%$)* is accepted and considered as delivered. The same criterion is adopted for evaluation and acceptance of the final product (software completed and delivered).

Artifacts and Tasks

Specifically for this work we adopt the following definitions for the terms artifacts and tasks: (1) **Artifacts:** parts of the product to be developed. It consists of tasks of the same group or type. The artifacts defined are: Architecture, Codes, Database, Documentation and Test. and (2) **Tasks:** Set of 26 items (jobs) to be developed, divided into categories/types that make up the artifacts.

The complete list of tasks can be found in (Ribeiro, tese 2017).

Scheduling of Tasks

The production process and the scheduling of tasks in the production environment was performed by a model based on the Dynamic Job Shop Problem (DJSP) was developed with the following characteristics.

- A shop ***S*** is composed of a set ***L*** of production lines;
- A production line ***L*** is composed of M_i set of machines;
- Each M_i machine is represented by an individual ***i***;
- A task ***j*** is denoted by $j_{i \rightarrow n}$;
- A set of artifacts is denoted A_j
- An operation O_t is denoted by $M = \{M_i, A_j\}$, where M_i are machines at production line and A_j are artifacts for each task j_n in a time ***t***;
- Each shop $S \supset L_M$ components processing tasks;
- $D = \{A_1, A_2, ..., A_n\}$ are the sets of artifacts produced and delivered at a time ***T***.
- Each machine must produce and deliver a D_A set of artifacts at the end of each UP-Like phase.
- W_j is the set of predecessor tasks and must be scaled first;
- A task ***j*** can be rescheduled until it is completed or until it reaches the expected quality, regardless of the UP-Like phase.

A simple case of the Dynamic Job Shop (DJS) modeled for a production line *(L)* running with 2 machines $L=\{M_1, M_2\}$, processing 2 tasks *j*, staggered at a given time *T* in a phase of the UP-Like, can be represented by Figure 3.

Simplest case DJS environment: (1) $L = M_i A_j$; (2) $W= \{j_1, j_2\}$; (3) $T = T_{M_1} + T_{M_2}$; *and* (4) $O_t = T + L$

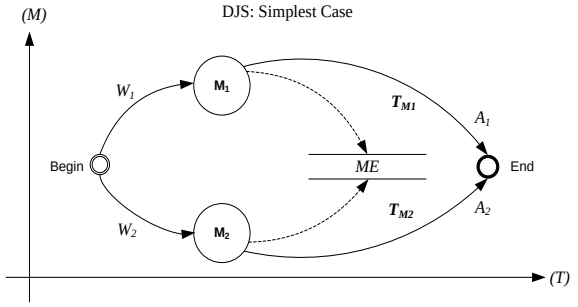

Figure 3. Simplest case (Dynamic Job Shop model)

Constraints Definition

There are two distinct sets of problems in SDP: (1) Qualitative constraints: Directly associated with problems who involve an individual or the development team. (2) Quantitative constraints (productive bottlenecks): problems that have impacted directly on the final product. Usually caused by a qualitative constraint.

Qualitative constraints are identified and handled at runtime. Quantitative constraints (bottlenecks) are problems detected after the data quantitative analysis, *i.e*, after the execution. This implies rescheduling the associated tasks in order to achieve the required completeness and new resources are used: time, peoples and computational tools.

Constraints Identification

For qualitative constraints we apply the following terms:

- Monitoring: continuous observation of development teams;
- Follow up: current status and project evolution;
- Meeting: can be of two types: (1) with the development teams; and (2) punctual meetings with a specific team member. Usually depends of observation on the recurring problem.
- Subjective analysis: partial analysis of the data (tasks produced) at runtime, such as: identification of a poorly constructed model; a bad code or a database with structural problems.

For quantitative constraints we apply the following variables:

- **Time:** calculation of time consumed by task/artifact. Time is measured by task, by set of artifact and by participant (individual) and by the team.
- **Quality:** calculation of the weighted average on the quality of the artifact delivered, according to "Measurement Criteria".

- **Survey**: The survey is an interview where participants point out the level of difficulty to complete an activity. Where: 0 = no difficulty; 1 = little difficulty; 2 = moderate difficulty; And 3 = too much difficulty. For the survey, a weighted average is also calculated where the weights are the own scores assigned by the participants in the tasks.

Treatment of qualitative constraints

The quantitative constraints were treated from the following items.

1. **Lectures:** The expository classes are preventive actions, *i.é*, are applied for leveling of knowledge and before the execution of a task or stage of development; In industry, expository classes can be replaced by training or instruction given by more qualified and experienced professionals.
2. **On-site visit:** the on-site visit is made to a team at runtime. The on-site visit is done both to identify and map a problem/constraint, and to treat it;
3. **Alignment meeting:** This is a meeting with the teams where the problem was identified. The goal is always to address the constraint and not point or accuse the source of the problem;
4. **Expositive Feedback:** is an exposition of the problem together with a guideline for a possible treatment and should be presented to all teams. The idea is to avoid that the same problem happens in the other teams, anticipating the treatment proactively;
5. **Written feedback:** is a written record that must be submitted to the team that shows a constraint.

Application of TOC

The logical foundation of TOC is based on the five steps that determine the concentration of the effort to improve the productive capacity of environment of development. TPC is a production scheduling method that has been developed to support this rationale and is intended to "protect" a Capacity Constrained Resource (CCR). The identification and treatment of the qualitative constraints proposed in this work goes through this arrangement to improve the productive capacity of the team member, especially when applied to TOC steps 1 and 4. Thus, the treatment is made with the pointing (identification) of the problems (constraints) and strictly follow the 5 steps of TOC.

1. **Identification** and evaluation of the constraint;
2. **Explore** the constraint to transform/modify this "weak link" of the process;
3. **Subordinate** other resources to the constraint, such as relocating the task to other team members;
4. **Elevate** the constraint, which in this case implies alternatives to reduce the impact of the constraint on the process and the product;
5. **Avoid inertia** by re-evaluating the process, the teams, the constraints found, to minimize the risk of arising new constraints.

QUALITATIVE ANALYSES

The qualitative evaluation applied in this research obeys the validity criteria with the use of neutrality, observation, long-lasting/on-site engagement, and especially with the generalizability and legitimacy of empirical study, as suggests Newman & Benz [14].

The search for evidence of bottlenecks in the software development environment started from the ethnographic observations noted in the register of the experiment.

Experimental Data

Table 1 presents the independent variables used in each experimental round (Exp 1. Exp 2 and Exp 3).

Id	Independent Variables	Values		
		Exp 1	***Exp 2***	***Exp 3***
1	Period of Execution	*08/12/2014 to 12/04/2014*	*03/17/2015 to 07/16/2015*	*10/13/2015 to 03/17/2016*
2	Duration (weeks)	*17*	*18*	*19*
3	Students Total	*35*	*32*	*18*
4	Total students (other areas)	*03*	*02*	*0*
5	Groups Total	*10 (9)*	*9 (8)*	*6 (4)*
6	Lectures Number	*10*	*09*	*06*
7	Practical Classes Number	*17*	*18*	*19*
8	Number of on-site visit	*4*	*4*	*4*
9	Total expositive feedback	*4*	*4*	*4*
10	Write feedback total	*4 p/LP*	*4 p/LP*	*4 p/LP*

Table 1. Independent variables and values of experiments

The total of groups in parentheses of Id 5 of table 5, represents the groups that concluded the experiment.
For each experiment a specific domain was used.

1. **Domain:** Rent a car control system (Exp 1);
2. **Domain:** Academic control system (Exp 2);
3. **Domain:** Ticket sales control system (Exp 3).

Qualitative Constraints Identified in PDS

The constraints found were separated into two groups: (1) behavioral restrictions; and (2) technical constraints.

Behavioral Constraints (BC)

The constraints in Table 2 represent the BC qualitative found in the shops of Experiments 1, 2 and 3.

Behavioral Constraints				
Id	***Constrains***	***Exp 1***	***Exp 2***	***Exp 3***
BC1	Internal communication	Yes	Yes	No
BC2	Internal Relationship (Conflicts)	Yes	Yes	Yes

BC3	Pro-activity and Passivity	Yes	Yes	Yes
BC4	Commitment to the Project	Yes	Yes	Yes
BC5	Focus Deviations: External impact of other curricular activities	Yes	No	No

Table 2. Behavioral constraints

Treatment of Behavioral Constraints (BC)

The treatments applied in the BC will be presented in the following topics. The topics will also present the groups (teams) and phase of the UP-Like in which the constraint occurred.

- **BC1:** constraint evidenced in the groups (G3 and G9) of Exp 1 and group (G2) of Exp 2. The constraint occurred in the following phases of the UP-Like: Exp 1 (3rd phase); Exp 2 (3rd and 4th phase).
  Treatment: (1) guidance: about the importance of establishing a solid channels of communication and follow-up these channels; (2) billing: rigidity in the evaluation of results; (3) alert: about the evaluation (score) of the software engineering discipline.
- **BC2:** constraint evidenced in the group (G3) of Exp 1, (G1) of Exp 2 and group (G2) of Exp 3. The constraint occurred in the following phases of the UP-Like: Exp 1 (3rd phase); Exp 2 (2nd, 3rd and 4th phase) and Exp 3 (3rd phase).
  Treatment: in the specific case of this study, the problem was mediated by the interlocutor, with the following actions: (1) private conversation with the team; (2) individualized conversation with those involved; (3) guidance and motivation regarding conflict resolution; (4) re-presentation of the importance of the project and the common objective of the team; and (5) threat of penalties: all group would be penalized for not complying with the project stages.
- **BC3:** constraint evidenced in the groups (G2, G3, G7 and G8) of Exp 1, group (G4, G8 and G9) of Exp 2 and group (G1 and G4) of Exp 3. In the three cases (Exp 1, Exp 2 and Exp 3). The constraint was observed after the 2nd phase of process.
  Treatment: (1) redistribution equitable of tasks; (2) motivation to the passive team members; and (3) billing on passives about the work plan.

  BC3 was identified with a constraint because the team members pro-active assume tasks beyond their productive capacity, and this has a direct impact on development.
- **BC4:** constraint evidenced in the group (G3) of Exp 1, group (G1) of Exp 2 and group (G2) of Exp 3. The constraint was observed after the 2nd phase of process UP-Like.
  Treatment: (1) continuous billing of the groups; and (2) instruction on the importance of the work.
- **BC5:** constraint evidenced in all groups of Exp 1, Mainly (G3), and group (G2) of Exp 2. The constraint occurred in 2nd e 3rd phases of both experiments (Exp 1 and Exp 2).
  Treatment: (1) adjustment of the schedule of deliveries according to the academic calendar; and (2) postponement: the delivery was postponed in one week.

Technical Constraints (TC)

Table 3 presents the TC (qualitative) found in the shops of Experiments 1, 2 and 3.

Technical Constraints				
Id	***Constraints***	***Exp 1***	***Exp 2***	***Exp 3***
TC1	Maturity on the domain	Yes	Yes	Yes
TC2	System documentation	Yes	Yes	Yes
TC3	UML language: Knowledge and correct notational use	Yes	Yes	Yes
TC4	Tools used: Knowledge and ability	Yes	Yes	Yes
TC5	Unified Process	Yes	Yes	Yes
TC6	Architectural model (MVC)	Yes	Yes	Yes
TC7	Test strategies	Yes	Yes	Yes

Table 3. Technical constraints

Treatment of Technical Constraints (TC)

The treatments applied in the TC will be presented in the following topics. The structure adopted presents the same standard shown in the previous section.

- **TC1:** constraint evidenced in all groups of Exp 1, except (G8) and (G2), in the group (G4) in Exp 2 and group (G2) in the Exp 3. In the three experiments, the constraints occurred in the phases 1 and 2 of UP-Like.
  Treatment: elaboration of a pro-analysis and a pro-specification formal and direct of the requirements, containing: (1) identification of the main functionalities and classes of the system; (2) clarification of generic doubts (for all groups – classroom lecture); and (3) clarification of specific doubts (for each group - on-site visit).
- **TC2:** constraint evidenced in all groups of the three experiments. The constraint occurred in the phases 1, 2 and 3 of all experiments.
  Treatment: (1) preparation of a road map; and (2) lectures explaining how to build the documentation.
- **TC3:** constraint evidenced in all groups of Exp 1 and Exp 2, except (G8) from Exp 1 and (G4) from Exp 2. In the Exp 3 the groups (G1) and (G2). The constraint was observed in the 1st and 2nd phases of the Exp 1and Exp 2 and 1st, 2nd and 3rd phases of Exp 3.
- **Treatment:** (1) application of five theoretical lectures on UML involving: UML elements and the diagrams defined in the project to compose the software plant.; and (2) a case study of a similar domain was shown to

increase the capability and ability to abstraction of the requirements.

- **TC4:** constraint evidenced in all groups of Exp 1, all groups of Exp 2, except group (G4) and group (G2) in the Exp 3. The constraint was observed in the 1st, 2nd and 3rd phases in all experiments (Exp 1, Exp 2 and Exp 3).
  Treatment: (1) Attempt to level the groups with specific instructions on each front of work; (2) incentive to explore the individual abilities of each team member; and (3) Guidance on internal redistributing of tasks according to individual skills and knowledge.
- **TC5:** constraint evidenced in the all groups of all experiments. The constraint was observed in the 1st and 2nd phases in all experiments.
  Treatment: (1) transmission of knowledge through lectures and practices; and (2) follow-up during on-site visits.
- **TC6:** constraint evidenced in all groups of Exp 1, except (G2, G8, and G7); and all groups of Exp 2, except G4 and the group (G1) of Exp 3. The constraint was observed in the 2nd, 3rd and 4th phases of the UP-Like.
  Treatment: (1) lectures: expositive class; (2) presentation of models and examples *in-loco* for groups with difficulty; (3) technical explanation of the model and its correct use; and (4) mandatory modeling of MVC with UML sequence diagram for a better understanding of logic model.
- **TC7:** constraint evidenced in all groups of Exp 1, except group (G8); all groups of Exp 2 and groups (G2, G5) in Exp 3. The constraint was observed in the 3rd and 4th phases of (Exp 1 and Exp 2) and phases 2nd, 3rd and 4th in Exp 3.
  Treatment: (1) lectures on software tests, especially on V & V test, unit test and functional test. (2) presentation of examples *in loco* for the groups with difficulty in carrying out the tests; (3) technical explanation on the correct application of tests.

Some TC identified may not be found in processes performed in the software industry, because in general, the development teams in this environment are formed by qualified and experienced professionals.

QUANTITATIVE ANALYSES

Quantitative research fits into the category of empirical studies or statistical studies, which includes: experimental studies, quasi-experiments, pre-tests, and post-tests [10]. According to Harwell [10], the objective of quantitative methods is the maximization, replicability, generalization of the results and the prediction of the researcher in relation to their experiences, perceptions and prejudices.

Quantitative analysis will be used in this work to understand the problems encountered in the SDP and to validate the identified bottlenecks. The quantitative analysis will also allow us to verify if the qualitative constraints reflect on productive bottlenecks in the software development process.

Quantitative Variables Analyzed

To measure quantitatively we use three dependent variables: (1) Effort**:** **t**ime consumed; (2) Quality**:** completeness and correctness; and (3) Difficulty**:** survey on the difficulties perceived by the members of each team**.**

Effort Measured

Team effort is calculated from the number of hours worked in each artifact. This is an important variable because it allows visualizing anomalies in the development process once a task that consumes a large number of hours has an indication of the existence of qualitative constraints that possibly going to cause a bottleneck in the productive process.

Owing the limitation of the pages of this work, only the quantitative data of the 3rd experiment (Exp 3) will be presented. The complete analysis of the "*quanti*" data can be found in the work of [19].

Quantitative Analysis of Effort

The first step in the quantitative analysis is to check the spent time by teams to construct each artifact. Figure 4 shows the total time consumption (in minutes) of each group.

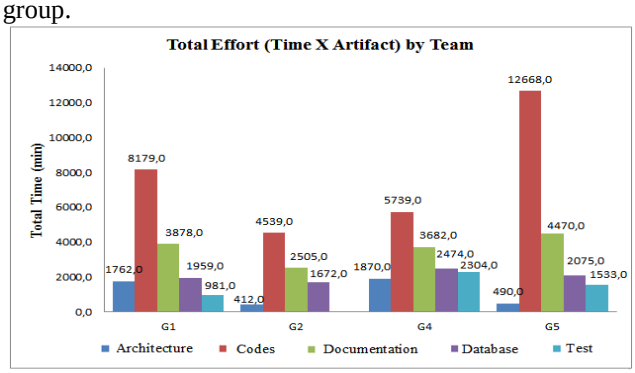

Figure 4. Total time consumed by team of experiment 3

In fact, the activities that involve the creation of codes are the most time consuming, including in the less efficient teams.

Another interesting data is the fact that artifacts time-consuming often involve more people to build. Table 4 shows the total time spent by each component *(A, B, C and D)* of the development teams. Where is possible to observe that the "*Codes*" and "*Documentation*" artifacts besides consuming more time, also involves practically the whole team. In the first analysis this does not show a productive bottleneck, but it is a relevant factor that must be carefully analyzed.

Team		Architecture	Codes	Documentation	Database	Test
G1	*A*	*367*	*2517*	*1486*	*436*	*245*
	B	*338*	*2159*	*400*	*549*	*736*
	C	*533*	*2814*	*967*	*395*	*0*
	D	*524*	*689*	*1025*	*579*	*0*
Total		**1762**	**8179**	**3878**	**1959**	**981**

G2	*A*	*292*	*785*	*1092*	*822*	*0*
	B	*120*	*650*	*1310*	*10*	*0*
	C	*0*	*187*	*103*	*0*	*0*
	D	*0*	*2917*	*0*	*840*	*0*
Total		**412**	**4539**	**2505**	**1672**	**0**
G4	*A*	*160*	*390*	*2887*	*120*	*660*
	B	*40*	*2475*	*618*	*570*	*360*
	C	*1670*	*2874*	*177*	*1784*	*1284*
Total		**1870**	**5739**	**3682**	**2474**	**2304**
G5	*A*	*120*	*2861*	*2040*	*1256*	*1124*
	B	*370*	*9807*	*1297*	*594*	*0*
	C	*0*	*0*	*1133*	*225*	*409*
Total		**490**	**12668**	**4470**	**2075**	**1533**

Table 4. Total time spent by team members

Quality Measured

To define a factor to measure the quality of the artifacts in each delivery, we created a scalar order of 0.0, 0.5, and 1.0.

We also use the same criterion for the overall assessment, *i.e*, the average value of the product delivered by adding the five artifacts. In order to quantify the quality of deliverables, we use this same scale, considering only (0 and 1), being "*0"* for undelivered or partial delivered artifact and "*1"* for artifact delivered (complete). This is interesting because it allows the numerical visualization of the production of each group for each delivery and by artifact.

We also established an alphabetical order to arrange the artifacts and consequently the assigned weight to generate the corresponding binary number in each delivery.

Table 5 present the binary chain and decimal factor used to measure the quality pf artifacts. Figure 5 presents two graphics: (a) shows the evolution of the deliveries according to the factor obtained by Table 5; and (b) shows the percentage delivered by the groups in each phase of the UP-Like.

Factor to Measure Quality (Exp 3)																							
Group	Inception (Delivery 1)						Elaboration (Delivery 2)						Construction (Delivery 3)						Transition (Delivery 4)					
	Architecture	Database	Codes	Documentation	Test	Decimal factor	Architecture	Database	Codes	Documentation	Test	Decimal factor	Architecture	Database	Codes	Documentation	Test	Decimal factor	Architecture	Database	Codes	Documentation	Test	Decimal factor
G1	*1*	*1*	*1*	*0*	1	29	*0*	*1*	*1*	*0*	*1*	13	*1*	*1*	*1*	*1*	*1*	31	*1*	*1*	*1*	*1*	*1*	31
G2	*0*	*1*	*0*	*0*	1	9	*0*	*1*	*0*	*0*	*1*	9	*0*	*0*	*0*	*1*	*0*	0	*0*	*1*	*0*	*0*	*0*	8
G4	*1*	*1*	*1*	*1*	1	31	*0*	*1*	*0*	*0*	*1*	9	*1*	*1*	*1*	*1*	*1*	31	*1*	*1*	*0*	*1*	*1*	27
G5	*1*	*1*	*1*	*1*	1	31	*1*	*1*	*1*	*1*	*1*	31	*0*	*1*	*0*	*1*	*0*	0	*1*	*1*	*0*	*1*	*0*	26

Table 5. Factor of convers of quality

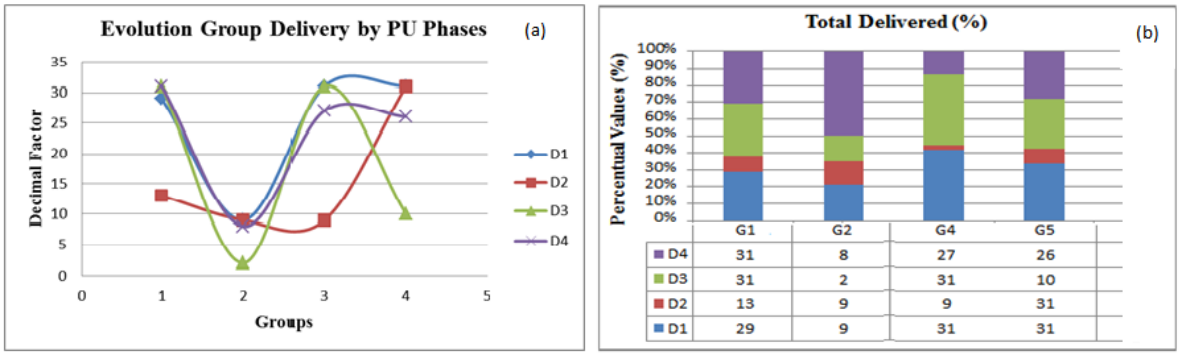

Figure 5. (a) Evolution of Delivery by Groups (b) Percentage delivered by UP-like phases

Based on the numerical analysis of the Table 5 and the graphical visualization of the Figure 5 (a) and (b), is possible to notice the difference in the productivity of each group in each delivery. The dispersion shown in figure 5 shows the evolution of deliveries, with delivery 2 (elaboration) being more problematic. This happened for three reasons: (1) volume of task in the phase 2; (2) type of tasks entered in the shop at this stage; and (3) deadline syndrome [22] due to the prioritization of external activities.

It's also possible to observe that the two most time consuming tasks, (1) coding: presented low quality in the groups (G2, G4 and G5), especially in the delivery of phase 4; And (2) Documentation: presented low quality in the groups: (G1) specifically in phase 1 and 2; And (G2) in the 4 phases of the process. The reasons for this may be the qualitative characteristics of members of the development teams.

Difficulty Measured

To measure the difficulty of the groups in performing the tasks, a survey was elaborated with the following factors and weights: 0-No difficulty, 1-few difficulty; 2-Moderate difficulty; and 3-very much difficulty. The survey was applied at the end of the execution of each UP-Like phase. And the members were told to respond only in what they participated.

Table 6 presents the weighted average of the Survey *(μS)*, obtained by each team/person.

Survey: Difficulty Presented by Groups					
Grupos	Architecture	Codes	Documentation	Database	Test
G1	0,531	1,047	1,111	0,790	0,753
G2	1,377	1,839	1,216	1,613	0,868
G4	1,250	1,438	1,125	1,063	1,014
G5	1,227	1,378	1,396	0,979	1,375

Table 6. Difficulty presented by groups

Figure 6 presents the results measured with the survey.

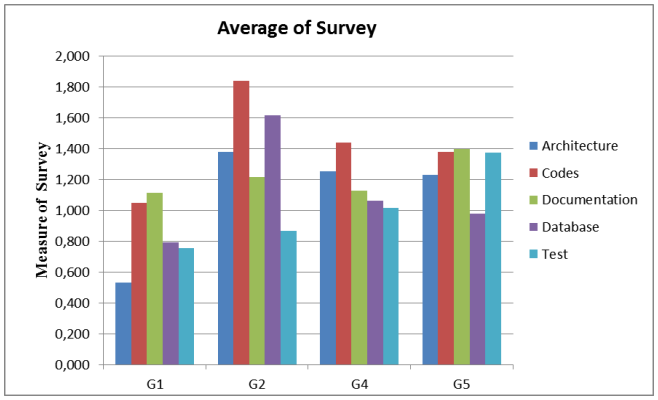

Figure 6. Survey Measured

The perception of difficulty of the teams elected the artifacts (Codes, Architecture and Documentation) as the most difficult. Group (G2) also presented great difficulty with database.

Descriptive Analysis

Clearly, after analyzing these three variables it is noticeable that the activities that involve documentation, architectural model and coding are the tasks that most require of the development teams. We can consider that the construction of these artifacts has restrictive elements that prevent the maximization of production in the PDS. It is also noticeable that these quantitative results have a direct correlation with the qualitative results, given the technical and behavioral characteristics of the people who work in software construction.

Further Work

In future works, we intend to analyze which tasks that make up the artifacts that really impact development, using the same measurement criteria, but applied to specific tasks that make up the artifacts analyzed here. Another front that can be explored in the future is the application of this experiment in the software industry to verify if the problems found are the same or if it has some correlation with the experiment realized in the academy. Thus we intend to typify and validate the bottlenecks of the software development process.

ACKNOWLEDGMENT

Sincere thanks to the students of the Department of Computer Science at Federal University of Rio de Janeiro, Fundamentals Software Engineering, class (2014/2, 2015/1 and 2015/2) that agreed and collaborated with this research.

CONCLUSION

For decades, the manufacturing industry has carried out studies to identify bottlenecks in production processes, especially with the application of TOC. In this aspect, the software production still walks by very slow steps. Identifying bottlenecks in the PDS, in fact, is a research opportunity that must be better explored in order to reduce the impacts on the process and consequently the cost of projects of the software construction.

Although this study was carried out in a software engineering learning environment, we believe that similar studies can also be applied in the software industry. To this end, some adjustments to the problem domain, the development team, the SDP, and cultural and regional aspects may be required.

The realization of this experimentation in an industrial environment (real / professional) is a limitation of this work that we intended to be carried out in the future. Because this type of study can validate the evidences found in this research or even generate new discoveries.

REFERENCES

1. Steve Adolph, Wendy Hall and Philippe Kruchten. (2011) Using grounded theory to study the experience of software development; Empirical Software Engineering; Electrical and Computer Engineering – University of British Columbia Vancouver Canada; August, 2011, Vol. 16, Issue 4, pp. 487–513; Vancouver – CA.
2. Raymond P. L. Buse, Thomas Zimmermann. (2012) Information needs for software development analytics; Proceeding; ICSE '12 Proceedings of the 34th International Conference on Software Engineering; IEEE Press Piscataway, NJ, USA ©2012 ; Pages 987-996; Zurich, Switzerland — June 02 - 09, 2012 .
3. Jeffrey C. Carver, Letizia Jaccheri, Sandro Morasca (2003) Issues in Using Students in Empirical Studies in Software Engineering Education. Proceedings of the Ninth International Software Metrics Symposium (METRICS'03) 2003; IEEE Computer Society;
4. Paul Clarke and Rory V. O'Connor. (2012) The situational factors that affect the software development process: Towards a comprehensive reference framework; Information and Software Technology, Volume 54, Issue 5, May 2012, Pages 433–447

5. Alfred S. Goldhaber, Michael M. Nieto (2010) "Photon and graviton mass limits", Rev. Mod. Phys. (American Physical Society); RevModPhys; 2010.Gupta, A. Bhardwaj, A. Kanda (2010) Fundamental Concepts of Theory of Constraints: An Emerging Philosophy; World Academy of Science, Engineering and Technology 2010.
6. Elijah M. Goldratt, (2002) The Critical Chain; North River Press; 2002; Reprint and Translated; Nobel Press, São Paulo – SP; Brazil, 2006.Goldratt, E. M.; It's Not Luck, Ed. The North River Press, 1993; Paperback Reprint 2002.
7. Elijah M. Goldratt, Jeff Cox (2006) The Goal: A Process of Ongoing Improvement; 2ª. Ed.; Nobel Press; São Paulo – SP; Brazil; 2003; Reprint 2006.
8. Georgios Gousios, Martin Pinzger and Arie van Deursen. (2014) An exploratory study of the pull-based software development model; Proceeding ICSE 2014 Proceedings of the 36th International Conference on Software Engineering; Pages 345-355; Hyderabad, India — May 31 - June 07, 2014; ACM New York, NY, USA ©2014.
9. Ajay Gupta, Arun Kanda, Arvind Bhardwaj (2010) Fundamental Concepts of Theory of Constraints: An Emerging Philosophy; World Academy of Science, Engineering and Technology 2010.
10. Michael R. Harwell, (2011) M.R. Research design in Qualitative, quantitative, and mixed methods (Chap 10) in The Sage handbook for research in education: Pursuing ideas as the keystone of exemplary inquiry; 2nd *Edition; SAGE Publications; Thousand Oaks, CA;* 2011.
11. Seonmin Kim, Victoria J. Mabin, John Davies (2008) The Theory of Constraints Thinking Processes: retrospect and prospect; International Journal of Operations & Production Management; Vol. 28, Nr. 2 pp. 155-184, Emerald Group Publishing Limited; Wellington, New Zealand; 2008.
12. Gwanhoo Lee and Weidong Xia. (2010) Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility; *MIS Quarterly;* Vol. 34, No. 1 (March 2010), pp. 87-114
13. Laurie McLeod and Stephen G. MacDonell. (2011) Factors that affect software systems development project outcomes: A survey of research; The ACM Computing Surveys (CSUR) Journal; Volume 43 Issue 4, October 2011 ; Article No. 24; ACM New York, NY, USA.
14. Isadore Newman, Carolyne R. Benz (1998) Qualitative-Quantitative Research Methodology: Exploring the Interactive Continuum; Southern Illinois University Press; Carbondale - IL; 1998.
15. Agneta Nilsson, Laura M. Castro, Samuel Rivas and Thomas Arts (2016) Assessing the effects of introducing a new software development process: a methodological description International Journal on Software Tools for Technology Transfer February 2015, Volume 17, Issue 1, pp 1–16; Springer Link; 2016.
16. Shamuvel V. Pandit, Girish R. Naik (2009) Application Of Theory Of Constraints On Scheduling Of Drum-Buffer-Rope System; Second International Conference on Emerging Trends in Engineering (SICETE); IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE), 2009.
17. Shams-ur Rahman (1998) Theory of Constraints: A review of the philosophy and its applications, Perth, Australia International Journal of Operations e Production Management; vol.08 nr.04 pp336-355 1998.
18. Austen Rainer and Tracy Hall. (2013) A quantitative and qualitative analysis of factors affecting software processes, Journal of Systems and Software - Elsevier; Volume 66, Issue 1, 15 April 2003, Pages 7–21.
19. Sildenir A. Ribeiro (2017) Bottlenecks Identification in Software Development Process: A Proposal Based on the Principles of the Theory of Constrains; Doctoral Thesis; The Tércio Pacitti Institute; PPGI-Post Graduate Program in Informatics; UFRJ; Rio de Janeiro/RJ – Brazil; 2017.
20. Sildenir A. Ribeiro, Eber A. Schmitz and Antônio Juarez A. S. M. de Alencar. (2015) Bottleneck Identification in Software Development Processes: A Proposal Based on the Principles of the Theory of Constraints; Proceedings of 2015 IEEE 10th International Conference on Global Software Engineering (ICGSE 2015).
21. Sildenir A. Ribeiro, Eber A. Schmitz, Antonio J. S. M. Alencar, Monica F. da Silva (2017) Research Opportunities on the Application of the Theory of Constraints to Software Process Development; Journal of Software vol. 12, no. 4, pp. 227-239, 2017.
22. Sildenir A. Ribeiro, Eber A. Schmitz, Antonio J. S. M. Alencar, Monica F. da Silva (2017) The Deadline Syndrome: Origin, Causes and Implications in the Software Development Process; vol. 10, No. 2, pp.30-47; ISYS-Brazilian Journal of Information Systems - SBC-Brazilian Computer Society; Rio de Janeiro-RJ; june-2017.
23. Eli Schragenheim, H. William Dettmer (2000) A Whole System Approach to High Velocity Manufacturing: Simplified Drum-Buffer-Rope; Optimizing Supply Chain Business Performance. Boca Raton, FL: St. Lucie Press, 2000.
24. Sankar Sengupta, Kanchan Das, Robert P. Vantil (2008). A New Method for Bottleneck Detection; Proceedings of the 2008 Winter Simulation Conference. IEEE Xplorer; 2008.
25. Claes Wohlin; Per Runeson; Martin Höst, Magnus Ohlsson; Björn Regnell, Annika Wesslén (2012) Experimentation in Software Engineering; Springer-Verlag Berlin Heidelberg , 2012.