Software Engineering & Business Process Management: Interpretatio, Imitatio, Aemulatio

Prof.dr.ir. Hajo Reijers

1st IEEE International Workshop on Communicating Business Process and Software Models, September 23rd 2013
Some context


Pindarus (522 BC – 443 BC)  Horatius (65 BC – 8 BC)  Homerus (around 850 BC)  Vergilius (70 BC – 19 BC)
Software programs & business processes

- Design-time vs. run-time
- Information processing
- Compositional structure
- Humans in vs. out-of-the-loop
- Artifacts: design, maintenance, sense-making

Some reflections

Software engineering

```plaintext
1 6 K=1
2 6 IF (K=EQ.11) GO TO 8
3 6 READ,I,J
4 6 IF (J.GT.I) GO TO 65
5 6 GO TO 66
6 65 WRITE(6,6002)J,I
7 6002 FORMAT(' ',I3,' IS GREATER THAN ',I3)
8 6 K=K+1
9 6 GO TO 6
10 66 WRITE(6,6001)I,J
11 6001 FORMAT(' ',I3,' IS GREATER THAN ',I3)
12 6 K=K+1
13 6 GO TO 6
14 8 CALL EXIT
15 END
```

Business process management

Note: in neither domain are models the only or even the main artifact.
Design principles
Design principles

Loose coupling, tight cohesion
Law of Demeter:

1. Your method can call other methods in its class directly.
2. Your method can call methods on its own fields directly (but not on the fields' fields).
3. When your method takes parameters, your method can call methods on those parameters directly.
4. When your method creates local objects, that method can call methods on the local objects.

Translated:

1. You can play with yourself.
2. You can play with your own toys (but you can't take them apart).
3. You can play with toys that were given to you.
4. And you can play with toys you've made yourself.

http://c2.com/cgi/wiki?LawOfDemeter
Evaluating workflow process designs using cohesion and coupling metrics

Irene Vanderfeesten *, Hajo A. Reijers, Wil M.P. van der Aalst

Technische Universiteit Eindhoven, Department of Technology Management, PO Box 513, NL-5600 MB Eindhoven, The Netherlands
Received 9 January 2007; received in revised form 5 December 2007; accepted 13 December 2007
Available online 4 March 2008

Abstract

Building on the similarities between software programs and workflow processes, this paper proposes a heuristic that offers guidance for the creation and evaluation of process designs in administrative settings. Designers can use this heuristic to select from several alternatives the process design that is strongly cohesive and weakly coupled. It is argued that such a design will result in fewer errors during information exchanges and in more understandable activity descriptions. The paper includes an application of the heuristic in an industrial workflow setting, which supports its feasibility and practical value. The paper also presents the freely available CoCoFlow tool that implements the heuristic and its associated metrics.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Business process modelling; Workflow management; Quality metrics
Design principles
Design principles

Using Software Quality Characteristics to Measure Business Process Quality

A. Sclicci Gascioglu and Umut Demirci
Informatics Institute, Middle East Technical University, Istanbul, Turkey
+90 (312) 330 3741
as-sclicci@metu.edu.tr
demirci@metu.edu.tr

Abstract. Organizations frequently use product based organizational performance models to measure the factors of information system (IS) on their organizations. This paper introduces a complementary process based approach that is focused on measuring business process quality attributes. These quality attributes are defined on the basis of ISO/IEC 9126 Software Product Quality Model. The new process quality attributes are applied at an experiment and results are discussed in the paper.

1 Introduction

IS capabilities have been advancing at a rapid rate and motivating organizations to accomplish much more investment. In 2002, $788 billion was spent for IS in the United States alone [1]. Although IS expenditures seem quite high, there are few systematic guidelines to measure the organizational impact of IS investments [2], [3]. The available studies on organizational impact of IS focus on the product based organizational performance models to manage IS investment. These studies provide organizations with guidelines for measuring cost and time related issues, but they have some constraints in identifying IS effects, isolating the contributions of IS effects from other contributors and using the performance measures in specific categories of organizations such as public organizations.

In this paper, a complementary process-based approach is developed to measure the effects of IS on business process. This new approach focuses on the quality aspects of the processes. As it is known that business processes are one of the most fundamental assets of organizations, modifications performed on them whether in the way of improvements or innovations cause immediate effects on the success of the organizations. This approach therefore enables organizations to get early feedback about a potential IS investment. In the remaining charts of the paper firstly, related search is summarized as a background to depict the relation of our model to the IS literature. Secondly, the new model is introduced and its measurement categories are given. Thridly, implementation of the model and its results are summarized. Finally, conclusions and future works are stated.

2 Related Research

One of the most widely known models for measuring the effects of IS is DeLone and McLean IS Success Model [2]. The available studies in Organizational Impact W.A.P. van der Aalst et al. (Eds.): BPM 2005, LNCS 3469, pp. 274–379, 2005.

Analysis and Validation of Control-Flow Complexity Measures with BPMN Process Models

Elvira Robles1, Jorge Cardoso2, Félix García2, Francisco Ruiz3, and Mario Piattini3

1 Autonomous University of Tamaúlpas – FEANS
University Center Tampico-Madero SN, Tamaúlpas, México erolito@sust.edu.mx
2 SAP Research, Germany and University of Coimbra, Portugal jorge.cardoso@sap.com
3 Almeida Research Group, University of Castilla La Mancha Paso de la Universidad No. 4, 13071 Ciudad Real, Spain felix.garcia, francisco.ruizg, mario.piattini1@ucm.es

Abstract. Evaluating the complexity of business processes during the early stages of their development, primarily during the process modelling phase, provides organizations and stakeholders with process models which are easier to understand and easier to maintain. This presents advantages when carrying out evolution tasks in process models – key activities, given the current competitive market. In this work, we present the use and validation of the CPC metric to evaluate the complexity of business processes modelled with BPMN. The complexity of processes is evaluated from a control-flow perspective. An empirical evaluation has been carried out in order to demonstrate that the CPC metric can be useful when applied to BPMN models, providing information about their ease of maintenance.

Keywords: Business process models, BPMN, measurement, validation.

1 Introduction

Business process modelling is the first step towards the achievement of organizational goals, because its importance resides not only on the description of the process, but in that it also usually represents a preparatory phase for activities such as business process improvement, business process reengineering, technology transfer and process standardization [1].

But in all these activities the business process models are managed by different stakeholders (business process analysts, domain experts, technical analysts, software developers, among others). Therefore, one of their main purposes is support communication between stakeholders, and to fulfill this purpose business process models should be easy to understand and easy to maintain. High complexity in a process has several undesirable drawbacks: it may result in bad understandability, errors, defects, and exceptions, thus leading to the need for more time to develop, test and maintain the processes. Therefore, the first step towards reducing the complexity of processes is to first recognize its existence, and then, measure it.

© Springer-Verlag Berlin Heidelberg 2009
Design principles

Analysis: IBM and Pennsylvania share blame in massive IT train wreck

*Summary*: A huge IT failure illustrates problems that arise when customer and system integrator do not fulfill their responsibilities and obligations.

By Michael Krigsman for Beyond IT Failure | August 9, 2013 -- 12:12 GMT (03:12 PDT)

After significant delays and cost overruns, Pennsylvania has terminated a contract with IBM to provide new software for the state’s unemployment compensation system.
“Cyclomatic complexity (CC), another widely used industry indicator of software quality, represents the number of independent paths through the program source code. The CC number for individual modules or units can be calculated using one of several commercial tools. Best industry practice seeks to limit the complexity of individual code units to 10 or less because software modules having a CC greater than 10 have a higher risk of defects and a higher risk of "bad fixes" in which attempts to correct a defect result in new defects being created. Typically, CC values exceeding 20 should be considered alarming. While the percent of UCMS software modules with a CC greater than 10 is small, about 1,600 modules have a CC value greater than 20 which indicates a moderate to high risk. Of those with a CC greater than 20, almost 100 have a value of 50 or greater which means that they are not testable in a practical sense. The discovery of defects in User Acceptance Test may be, at least in part, the result of the high CC factor in these modules.”

What is the scientific basis for this?

“CC can be said to have absolutely no explanatory power of its own”

Correctness
Correctness
Correctness

Ter vermindering van de hoeveelheid schrijfwerk ter regelmatig van bewijsverplichtingen en ten behoeve van een compacte presentatie van programma’s voeren we het afkortingsmechanisme in dat beeld staat onder de naam “programma-annotatie”. We illustreren het aan een voorbeeld.

```
let N : int
{k N > 0}
let a : int
{k b : int
  a, b := 0, N+1
  {invariant P: 0 < a < b ∧ a² ≤ N < b², zie Noot0.
   varianten functie: b − a}
  do while a + 1 < b
      {c : int
        c := (a + b) div 2
        {P ∧ a ≤ c < b, zie Noot1}
        if c * c ≤ N → a := c {P, zie Noot2}
        N < c * c → b := c {P, als Noot2}
        {P}
      }
  od
  {P ∧ a + 1 = b}
{k dus, a² ≤ N < (a+1)²}
```

Bovenstaand programma is vrij volledig geannoteerd. De volledigheid bestaat hierin dat voor elke constituerende statement zowel pre- als postconditie uit de tekst zijn
Correctness

A test team before computers were invented

Anyone found any bugs yet?

Nope

Nope

I haven't either

Andy Glover cartoontester.blogspot.com © 2012
Correctness
Correctness

- Option to complete: $\forall M \in R(N,[i]) \ [o] \in R(N,M)$.
- Proper completion: $\forall M \in R(N,[i]) \ (M \geq [o]) \Rightarrow (M = [o])$.
- No dead transitions: $\forall t \in T \ \exists M \in R(N,[i]) \ (N,M)[t]$.

Liveness and boundedness of short-circuited net

$\Leftrightarrow$

Soundness of net
Correctness

Correctness Criteria Based Problems

1) Changing the Past (CP)
   - I on S:
     - X
     - Y
     - A
     - B
     - C
     - Completed
   - Data

2) LT: Loop Tolerance
   - I on S:
     - Y
     - A
     - B [loop_back]
     - C
     - D

3) DS: Dangling States
   - I on S:
     - C activated or started?
     - A
     - B
     - C
     - D

Marking Adaptation Problems

4) OC: Order Changing Problem
   - I on S:
     - A
     - B
     - C
     - D
   - Parallelizing B and C

5) PI: Parallel Insertion Problem
   - I on S:
     - Y
     - A
     - B
     - C
     - D
   - Inserting parallel branch

---

Experimentation
The Case for Collaborative Programming

Team programming usually means coordinating efforts of individual programmers who divide up the programming tasks for a large, complex system. Collaborative programming is used here to mean two programmers working jointly on the same algorithm and code. Previous research indicates that student programmers working collaboratively outperformed individual programmers. A follow-up field experiment was conducted using experienced programmers who worked on a challenging problem important to their organization, in their own environments, and with their own equipment. To the surprise of the managers and participants, all the teams outperformed the individual programmers, enjoyed the problem-solving process more, and had greater confidence in their solutions.

Description of the Experiment

“Effective problem-solving performance” is defined in terms of (a) the readability of the proposed solution, that is, the degree to which the problem-solving strategy could be determined from the subject’s work; and (b) the functionality of the proposed solution, that is, the degree to which the strategy accomplishes the objectives stated in the problem description. The variables READABILITY and FUNCTIONALITY were defined accordingly. Readability is a component of overall performance that is particularly important for a programmer to use a reasonable strategy and to use programming language structures appropriately and yet fails to solve the problem (in the sense of generating the correct output). In such cases, the programmer may have misinterpreted the problem description or overlooked a critical component. Overall success on the problem-solving effort is measured by a variable SCORE, which is a simple sum of component variables READABILITY and FUNCTIONALITY. Based on previous results, four predictions were made: 1. Programmers working in pairs will produce more readable and functional solutions to a programming problem than will programmers working alone.

2. Groups will take less time on average to solve the problem than individuals working alone.

3. Programmers working in pairs will express higher levels of confidence about their work (CONFID) and enjoyment of the process (ENJOY) immediately following the problem-solving session than will programmers working alone. (Positive feelings about the problem-solving process can affect performance. These feelings were assessed immediately following the problem-solving session by a two-item questionnaire.)

4. Programmers with more years of experience will perform better than programmers with fewer years of experience.

Aside from the pairing, conditions, and materials used in the experiment, the same for both experimental (teams) and control (individual) groups. These subjects were 15 full-time system programmers from a program trading firm working on system maintenance of three Unix workstations and a large database running Sybase. They used
Experimentation

**Extreme programming explained: embrace change**
K. Beck, C. Andres - 2004 - dl.acm.org

... You will discover how to: Involve the whole team in XP style. Increase technical collaboration through pair programming and continuous integration. Reduce defects through developer testing. Align business and technical decisions through weekly and quarterly planning. ...
Geciteerd door 7200 Verwante artikelen Alle 18 versies Importeren in BibTeX Meer »

**Strengthening the case for pair programming**
L. Williams, R. Kessler, W. Cunningham... - Software, ... - 2000 - ieeexplore.ieee.org

Abstract The software industry has practiced pair programming (two programmers working side by side at one computer on the same problem) with great success for years, but people who haven't tried it often reject the idea as a waste of resources. The authors demonstrate ...
Geciteerd door 514 Verwante artikelen Alle 47 versies Importeren in BibTeX Meer »

**The costs and benefits of pair programming**
A. Cockburn, L. Williams - Extreme programming examined, 2000 - dsc.ufcg.edu.br

ABSTRACT Pair or collaborative programming is where two programmers develop software side by side at one computer. Using interviews and controlled experiments, the authors investigated the costs and benefits of pair programming. They found that for a ...
Geciteerd door 420 Verwante artikelen Alle 24 versies Importeren in BibTeX Meer »

**All I really need to know about pair programming I learned in kindergarten**
L. Williams, R. Kessler - Communications of the ACM, 2000 - dl.acm.org

Pair programming is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test. This method has been demonstrated to improve productivity and the quality of software ...
Geciteerd door 258 Verwante artikelen Alle 119 versies Importeren in BibTeX Meer »

**The effects of pair programming on performance in an introductory programming course**

Abstract The purpose of this study was to investigate the effects of pair-programming on student performance in an introductory programming class. Data was collected from approximately 600 students who either completed programming assignments in a ...
Geciteerd door 225 Verwante artikelen Alle 17 versies Importeren in BibTeX Meer »

**Improving the CS1 experience with pair programming**
N. Nagappan, L. Williams, M. Forzli, E. Wosbe... - ACM SIGCSE..., 2003 - dl.acm.org

Abstract Pair programming is a practice in which two programmers work collaboratively on one computer, on the same design, algorithm, or code. Prior research indicates that pair programmers produce higher quality code in essentially half the time taken by solo ...
Geciteerd door 198 Verwante artikelen Alle 13 versies Importeren in BibTeX Meer »

**Pair programming illuminated**
L. Williams, R. Kessler - 2003 - books.google.com

--Written as instruction for pair programming newbies, and practical improvement tips for those experienced with the concept <p--Explores the operational aspects and unique fundamentals of pair programming; information such as furniture set-up, pair rotation, and ...
Geciteerd door 377 Verwante artikelen Alle 5 versies Importeren in BibTeX Meer »

**Experimental evaluation of pair programming**
L. Williams, L. Albinati - European Software Engineering Conference and Metadata, 2001 - Citeseer

...
Experimentation

- Factor and factor levels
- Subjects
- Objects
- Tasks
- Response variables
- Hypotheses
- Parameters
- Design
- Instrumentation
- Data collection
Experimentation

a randomized 2x2 factorial experiment

“Model interpretation occurs in two stages (Newell & Simon, 1972), these being perceptual processing (seeing) and cognitive processing (understanding). During perceptual processing, computational offloading effects can occur that reduce the processing burden of the cognitive system because the parsing of model elements to different semantic components is performed by the perceptual sensors (Treisman & Gelade, 1980). Several studies suggest that perceptual processing largely explains differences in the effectiveness of the subsequent cognitive processes (Larkin & Simon, 1987; Petre, 1995). These considerations suggest that the perceptual discriminability effect of an explicit visual syntax of Gateway constructs will be stronger when model complexity is increased.”

Summary

- Illustration of areas where:
  - SE clearly source of inspiration for BPM
  - SE and BPM go head-to-head
  - BPM improves upon SE

It sounds like a nice marriage.

Hajo Reijers
h.a.reijers@tue.nl
http://www.reijers.com
Twitter: @profBPM