
Manuel Mastrofini
Ph.D. Candidate

Giovanni Cantone
Advisor

Forrest Shull
Co-Advisor

Department of Computer Science and Civil Engineering

Ph.D. in Computer Science, Control and Geoinformation
Ph.D. Program Coordinator: Giovanni Schiavon

2010-2013
Page intentionally left blank.
Acknowledgements

At the very beginning, I thought it would have been fun writing this part of the thesis. Now that I am doing it, I actually believe it is a good thing that everyone who collaborated to this achievement be aware and can receive the due recognition. This way, everyone will share his or her own responsibilities and will not be allowed to complain that it does not work, it is not clear, it is boring and blablabla. However, do not try to get too much credit: I did it. It is my treasure. You just somehow contributed.

Also, you may remember that in my Master thesis, other than a few typos, I mentioned everyone, name by name. This time I decided to be slightly less explicit and much more concise, so let’s start.

Everything I have done, I dedicate it to my family. My extended family, the ones I love. I hope they are proud of me, as this would make me proud of myself. And happy.

I owe a profound and sincere gratitude to Prof. Giovanni Cantone, carrot and stick and continuous encouragement, as well as to Dr. Forrest Shull, illuminating guide.

I still strongly believe that, without friends, life would be very sad and ugly. I am lucky, because I have some of these friends that still walk on my side, as I walk on their side, so that we are not alone when it is stormy where we are.

As of today, challenges keep coming, but I am steady. So, please, stay with me, do not leave me alone, and I will keep being steady and will keep walking on your side. As a reward, I will thank you also in my next acknowledgment section (which is going to be something that you really do not expect) and I might even offer you some good cake or a beer, as soon as I manage to get some free time.

Thanks.

Science demands nothing less than the fervent and unconditional dedication of our entire lives.
Table of Contents

Abstract ..................................................................................................................................................'7
Structure of the thesis .......................................................................................................................... 8

SECTION I: BEHIND THE CURTAIN

CHAPTER I. DECISION MAKERS IN THE FIELD OF SOFTWARE ENGINEERING: THE GPS METAPHOR ............ 10
CHAPTER II. THE ROLE OF EMPIRICAL SOFTWARE ENGINEERING ......................................................... 11
CHAPTER III. CONTRIBUTION OF THIS WORK: THE IMPACT VECTOR MODEL ................................................ 13
CHAPTER IV. CONCEPTUAL ARCHITECTURE OF THE MODEL ................................................................. 14

SECTION II: IMPACT VECTOR MODEL

CHAPTER V. SHAPING THE MODEL ........................................................................................................ 18
  V.I. ASSUMPTIONS .................................................................................................................................. 18
  V.II. CHARACTERISTICS OF THE MODEL ............................................................................................. 19
  V.III. TAXONOMY .................................................................................................................................. 19
CHAPTER VI. MATHEMATICAL FOUNDATION ......................................................................................... 20
  VI.I. FORMULATION ............................................................................................................................... 20
  VI.II. USING THE MODEL ....................................................................................................................... 22
CHAPTER VII. ADDRESSING THE OPEN PROBLEMS OF THE IMPACT VECTOR MODEL ......................... 25
  VII.I. ACCOUNTING FOR PARALLEL PROCESS ACTIVITIES .................................................................. 26
  VII.II. IMPACT VECTOR CONSTRAINTS .................................................................................................. 27
  VII.III. EMPIRICAL DEFINITION OF THE ACTIVITY PERFORMANCE FUNCTION AND COMBINATION FUNCTION ................................................................................................................................. 28

SECTION III: IMPACT VECTOR MODEL: LEVERAGING THE EXCELLENCE

CHAPTER VIII. WHERE DO IMPACT VECTORS COME FROM? ................................................................. 32
CHAPTER IX. ENABLING VALUE-BASED DECISIONS VIA THE IMPACT VECTOR MODEL ..................... 32
CHAPTER X. QIP AND GQM: CONTINUOUS PROCESS IMPROVEMENT VIA IMPACT VECTORS ............ 36
  X.I. QIP: QUALITY IMPROVEMENT PARADIGM .................................................................................. 36
  X.II. GQM+S: GOAL, QUESTION, METRIC + STRATEGIES .................................................................... 37
  X.III. MODELING CHOICES AND ALTERNATIVES WITH IMPACT VECTORS ....................................... 39
CHAPTER XI. IMPACT VECTOR MODEL: POINTS OF CONTACT WITH CMMI, ISO/IEC 15504 AND SIX SIGMA .... 41

SECTION IV: IMPACT VECTOR FRAMEWORK

CHAPTER XII. IMPACT VECTOR SCHEMA: ORGANIZE AND STORE IMPACT VECTORS ....................... 48
  XII.I. IMPACT VECTOR SCHEMA: MAIN REQUIREMENTS ....................................................................... 49
  XII.II. IMPACT VECTOR SCHEMA: ARCHITECTURE AND TECHNOLOGY ........................................... 50
### SECTION V: IMPACT VECTOR MODEL: 

**EXPERIMENTATION**

**CHAPTER XV.** IMPACT-ONE: VIRTUOUS COMPANY LEVERAGING THE IMPACT VECTOR FRAMEWORK ........ 58  
**XV.i.** FIRST STEPS FOR APPLYING THE IMPACT VECTOR MODEL: CONTEXT ........................................................................... 58  
**XV.ii.** FIRST STEPS: ACTIVITY IDENTIFICATION, DEFINITION, AND CHARACTERIZATION ........................................................................... 61  

**CHAPTER XVI.** TAILORING THE IBM RATIONAL HARMONY FOR EMBEDDED SOFTWARE .................. 63  
**XVI.i.** PROCESS DESCRIPTION ........................................................................................................................................... 63  
**XVI.ii.** ROLES IN OUR TAILORED HARMONY PROCESS .................................................................................................. 68  
**XVI.iii.** DESCRIPTION OF PROCESS ACTIVITIES ........................................................................................................... 70  
**XVI.iii.1.** Analyze Stakeholder Requests .......................................................................................................................... 70  
**XVI.iii.2.** Define System Validation Plan ........................................................................................................................... 72  
**XVI.iii.3.** Specify Requirements and Traceability to Stakeholder Requests ........................................................................... 73  
**XVI.iii.4.** Reviews and Risk Analysis ................................................................................................................................. 74  
**XVI.iii.5.** Review of Requirements ........................................................................................................................................ 75  
**XVI.iii.6.** Review of Black Box Use Case Design, System Test Plan, and Verification Reports ........................................... 75  
**XVI.iii.7.** Review of Architecture, White Box Use Case Design, System Integration Test Plan, and Reports .................. 76  
**XVI.iii.8.** Review Traceability Matrices .................................................................................................................................. 77  
**XVI.iii.9.** Analysis of Risks ...................................................................................................................................................... 78  
**XVI.iii.10.** Identify and Trace Use-cases to Requirements .................................................................................................... 78  
**XVI.iii.11.** Black Box Use Case Design .................................................................................................................................. 80  
**XVI.iii.12.** Define and Trace System, System Integration, and Unit Test Plans ........................................................................... 84  
**XVI.iii.13.** Verify Black-Box and White Box Use Case Design through Execution ............................................................................. 85  
**XVI.iii.14.** Define System Architecture ..................................................................................................................................... 87  
**XVI.iii.15.** White Box Use Case Design ................................................................................................................................... 90  
**XVI.iii.16.** Develop or Generate Units and Trace to White Box Use Case Design ............................................................................. 92  
**XVI.iii.17.** Perform Unit Test, System Integration Test, and System Test ................................................................................ 93  
**XVI.iii.18.** Perform System Validation ...................................................................................................................................... 93  
**XVI.iii.19.** Further Activities ..................................................................................................................................................... 94  
**XVI.iv.** ARTIFACTS AND TOOLS .......................................................................................................................................... 94  
**XVI.iv.1.** Checklists: a Support Tool for Reviews ................................................................................................................ 96  
**XVI.iv.2.** Contribution of Agile Models to the Proposed Development Process ............................................................................ 99  

**CHAPTER XVII.** TESTING ACTIVITIES: A COMPLETE EXAMPLE OF ACTIVITY CHARACTERIZATION ............. 107  
**XVIII.i.** SEQUENCING VERIFICATION TECHNIQUES: AN EXPERIMENT ................................................................. 115  
**XVIII.i.1.** Introduction ....................................................................................................................................................................... 115  
**XVIII.i.2.** Related Work ............................................................................................................................................................... 117
CHAPTER XXIV. TEST PERFORMANCE EVALUATION ................................................................. 128
  XIX.i. ANALYSIS OF RESULTS ............................................................................................ 129

CHAPTER XX. SELECTING TECHNOLOGIES FOR INCREASING PRODUCTIVITY: AN EXAMPLE WITH MOBILE TECHNOLOGIES 136
  XX.i. INTRODUCTION ........................................................................................................ 136
  XX.ii. PURPOSES OF THE STUDY AND RESEARCH METHODS ........................................ 137
  XX.iii. CHARACTERIZING THE TECHNOLOGIES TO STUDY ............................................. 138
  XX.iii.1. HTML5 + CSS3 + JavaScript (DT1) ................................................................. 140
  XX.iii.2. Platform Specific Software Development Kit (DT2,3) ......................................... 140
  XX.iii.3. Appcelerator® Titanium Mobile (1.8.1) (DT4) ...................................................... 140
  XX.iii.4. Adobe® Flex Mobile (4.5) + Adobe Air (DT5) ...................................................... 140
  XX.iii.5. PhoneGap (1.7.0) (DT6, 7, 8) .............................................................................. 140
  XX.iv. DRIVERS FOR TECHNOLOGY SELECTION ........................................................ 141
  XX.v. A MODEL FOR EVALUATING MOBILE TECHNOLOGY ........................................... 142
      XX.v.1. Implementation and Graphical View of the Model ............................................. 143
  XX.vi. THE PROCESS OF TECHNOLOGY DECISION MAKING FOR MOBILE .................. 146
  XX.vii. CASE STUDY ......................................................................................................... 147
  XX.viii. VALIDITY ISSUES, SOLUTION LIMITATIONS, AND LESSON LEARNED ............ 149

CHAPTER XXI. EXPLOITING SOFTWARE INSPECTION DATA FOR EARLY DEFECT REMOVAL ............ 150
  XXI.i. INSPECTIONS AT IMPACT-ONE: STATE OF THE PRACTICE .................................. 152
  XX.i.1. Decision Model: Statistical Analysis ................................................................... 156
      XX.i.1.1. Metrics .......................................................................................................... 156
      XX.i.2. Statistical Tests ................................................................................................. 156
  XX.i.3. DECISION MODEL: INTERPRETATION OF RESULTS ....................................... 158
  XX.i.4. THREATS ........................................................................................................... 160

CHAPTER XXII. OVERALL EXPERIMENTATION THREATS ....................................................... 161

SECTION VI: FINAL THOUGHTS AND FUTURE WORK

CHAPTER XXIII. FINAL THOUGHTS ..................................................................................... 164

CHAPTER XXIV. FUTURE WORK .......................................................................................... 164
   XXiv.i.1. Consolidating the Impact Vector Framework ..................................................... 164
   XXiv.i.2. Impact Vector Model and Empirical Software Engineering 2.0 ......................... 165
   XXiv.i.3. Business Analytics and the Impact Vector Framework ..................................... 165
   XXiv.i.4. Further Contexts of Application of the Impact Vector Framework .................. 166
Abstract

BACKGROUND. Software development is a mixture of technical, human and contextual factors, which make it complex and hard to manage. As in physics, medicine and social sciences, experimentation is the fundamental way for Software Engineering to understand, predict and cost-effectively control the software development process. In this thesis we deal with the improvement of the software development process, which may depend on a remarkable number of variables, even dozens; however, the currently available improvement models consider only small numbers of variables.

CONTEXT. The present work is a PhD thesis conceived and realized between 2009 and 2013 and developed partly in Italy and partly in the United States, with the contributions of the University of Rome Tor Vergata, the Fraunhofer Center for Experimental Software Engineering - Maryland, the National Aeronautics and Space Administration (NASA) and MBDA Italia.

GOAL. The purpose of this study is to understand, predict and control the software development process with focus on performance - modeled as a set of variables of the necessary cardinality, - from the point of view of decision makers, in the context of companies and agencies willing to achieve their business goals via a rigorous, predictable and engineered approach to software development. In other words, we aim to provide decision makers with a coherent and encompassing set of models and tools to support their Software Engineering job.

METHOD. In order to achieve this goal, we first need to define a theoretical foundation for our approach; then we need to validate such theoretical model, so to refine and complete its formulation. Subsequently, we need to enable decision makers to use and leverage the validated model in practice.

RESULTS. The main result of this thesis is a framework, tailored to both researchers’ and practitioners’ needs, to support the experimental investigation, and consequent improvement, of the software development process of organizations by detecting, modeling, and aggregating as many variables as necessary (up to ninety-four in our experience), and eventually use them to build measurement models. In this work, we report on the theoretical foundation of our framework and evidence of its successful preliminary validation on the field, with different organizations, including three major international companies, each having diverse goals and constraints in place. Specifically, the outputs of this study consist in a set of tailorable and usable methods to help experts make decisions concerning software development, along with a significant series of experiences to show its functionality and with some recommendations on how to turn the framework into practice.
Structure of the Thesis

In order to detail the roadmap to achieve the stated goal, this work has been structured as follow:

1. Section I tells the Global Positioning System (GPS) metaphor, i.e., a soft introduction to the standpoint the reader is expected to assume while approaching this work.
2. Section II provides the mathematical formulation of the Impact Vector Model, i.e. the model that we propose to enable meeting the stated goal. This mathematical formulation also includes definitions and assumptions which contextualize and scope the model. Eventually, some open problems of the model are pointed out and addressed.
3. Section III consolidates the model foundation by linking the Impact Vector Model to some pre-existing works which represent bright achievements in the history of Software Engineering. Such foundation includes: the concepts of the Value-Based Software Engineering; the Goal, Question, Metric plus Strategy paradigm; the Quality Improvement Paradigm; CMMI: ISO standards. Furthermore, a quick, but broad analysis of software (and system) development process models is provided.
4. Section IV packages the Impact Vector Model into a framework by providing a tool chain for supporting the model infusion in practice.
5. Section V focuses on the validation of the Impact Vector Model. We propose the application of impact vectors to a hypothetical “synthetic” company, which abstracts on and consolidates experiences that we gained by working with some real organizations worldwide across several years. In fact, data from trenches are precious and subject to restriction; so, we merge fiction and reality in such a synthetic company: while telling true stories of application of the Impact Vector Model, we subvert the outcomes of our studies. This way, we can illustrate quite real examples of application of our model, along with related limits and validity threats, while assuring the confidentiality of results and preserving sensitive data from being published.
6. Section VI reports some final thoughts and anticipates some future developments.

Since the structure of the thesis is top-down (first the context, second the theoretical model, and last model validation), a recommendation for the reader is not to linger on the formalisms of Section II, because a shallow understanding is enough to be able to easily follow and enjoy the next Sections. Eventually, a second reading of the theory will be faster and easier.
Section I: 
Behind the Curtain

«As our circle of knowledge expands, so does the circumference of darkness surrounding it».

Albert Einstein
Chapter I. Decision Makers in the Field of Software Engineering: the GPS Metaphor

In our daily lives, we use many tools and devices which support us in making choices. When such instruments are not available, we have to rely on our sensibility and intuition to undertake our decisions. An example of such instruments is a GPS (Global Positioning System) device with navigation software: given our current position and our destination, it can figure out the way to get there.

An interesting aspect of GPS devices is that, at the very beginning of the trip, they provide the driver with an estimation of distance and drive time. The distance is often very precise, as its calculation is based pretty much on certain data: satellite triangulation has a high resolution and, if the road model is accurate enough, then the computation of the distance is accurate as well. However, the time estimation is harder to provide. In fact, many variation factors can affect it, including: traffic, traffic lights, car characteristics, road conditions, driving style, stops, and unexpected events. Therefore, as long as you proceed in your trip, the time estimation may need to be adjusted.

Let us go a bit more in detail and give you a couple of examples. For a trip of 30 miles on the US-95 N/Veterans Memorial Highway, East of the Death Valley Park and 140 miles north of Las Vegas – basically, in the middle of nowhere –, the time estimation can be very accurate: factors like traffic, traffic lights and stops can be easily and reasonability ignored; therefore, assuming an average speed and that everything will be alright, the GPS makes life easy to a father answering his kid’s question: “are we there yet?”. The estimation of the GPS is very likely to be satisfying and appropriate: “30 minutes”.

I challenge you to use the same GPS, which worked perfectly in Nevada, for a trip of 10 miles on the crowded Great Ring Road around Rome – or any of your “favorite” roads you drive on every day to go to work – at around 8:30 PM. We do not want to put salt in your (and our) wound, but just point out that the estimation of the GPS can be highly ineffective in some conditions, if it accounts only for distance and average speed. In this situation, GPS devices should account for additional aspects, in particular real-time traffic condition, accidents and road work; and some GPS devices do, so that drivers can keep monitoring the trend of their trips and, maybe, they can notify their boss that they will be late for the planned meeting. So, the GPS may not fix the problem, but it can still be very useful to understand what is going on and, possibly, to take some precautions and countermeasures. Also, some GPS devices are adaptive and change the road accordingly to some conditions, so they can get even smarter and try to mitigate the problem, but we do not want to discuss this example any deeper.
Once that the GPS story has been told, let us jump on our beloved field of Software Engineering. During our experience, we realized that many large scale projects are much closer to the “Rome” example than to the “Death Valley” example. Uncountable factors must be considered by the decision makers, innumerable things can go wrong and accounting all possible attributes influencing the development is a utopia. However, if we are willing to accept to live with some degree of uncertainty, then we could be happy with having something like a GPS, providing some form of support to our development and some clue on what is happening with the project. Maybe, this sort of GPS for software development could be even capable to provide us with some advice on “road adjustments”.

Project management questions should be addressed by such hypothetical device, e.g.: “what testing technique is the most appropriate now, given my project status and goals?” or “can we do this task within the planned budget?” or “how can we assure that the Mars Rover will land the planet and will accomplish its mission with great success for the entire mankind?”. The answer to any of those questions can be very hard to provide and we do not want just a trivial “yes” or “very soon”, even though it could make everyone happy (for a while, at least).

How can we answer, then?

— Victor Robert Basili

Chapter II. The Role of Empirical Software Engineering

Empirical is an adjective denoting something derived or guided by experience, case studies or experiments. An empirical method, then, is an approach based on the collection of data to base a theory or derive scientifically valid conclusions. Also the notorious scientific method, by Galileo Galilei, founds on empirical methods and it is a stronghold in every field of science.

In the domain of Software Engineering (SE), the Empirical Software Engineering research field has emphasized the use of empirical methods (e.g. experiments, case studies, surveys and statistical analysis) to accumulate knowledge, in order to advance the state-of-art and the state-of-practice of SE.
In 1996, in the Editorial of the first volume of the “Empirical Software Engineering” journal, Victor R. Basili proposed a vision that, after 15 years of inspiration for great research in SE, also inspired the Impact Vector Model that will be presented in this work. Instead of proposing a synthesis of Basili’s vision, some excerpts are reported, which will settle the most appropriate standpoint for approaching next chapters.

“[...] physics, medicine, manufacturing. What do these fields have in common? They evolved as disciplines when they began learning by applying the cycle of observation, theory formulation, and experimentation. In most cases, they began with observation and the recording of what was observed in theories or specific models. They then evolved to manipulating the variables and studying the effects of change in the variables.

How does the paradigm differ for these fields? The differences lie in the objects they study, the properties of those objects, the properties of the systems that contain them and the relationship of the objects to the systems. So differences exist in how the theories are formulated, how models are built, and how studies are performed; often affecting the details of the research methods.

Software engineering has things in common with each of these other disciplines and several differences. [...] like other disciplines, software engineering requires the cycle of model building, experimentation, and learning; the belief that software engineering requires empirical study as one of its components.”

About 15 years later, the goal of predictable Software Engineering has clearly not been achieved yet and this work is a contribution towards that objective.

For skeptics and detractors of the feasibility of this goal, the following, additional thought, coming from the same source, might be illuminating.

“It is hard to build models and verify them via experiments - as with medicine. As with the other disciplines, there are a large number of variables that cause differences and their effects need to be studied and understood.”

Therefore, if we can attempt summarizing, SE is hard and complicated, but not more than many other sciences. And if we want to keep naming it “Engineering”, then we need to go ahead and build our models and theories, supported by experimentation, so to make software development more and more manageable.
Chapter III. Contribution of This Work: the Impact Vector Model

In very mature fields, scientists and engineers build models, according to some theories, in order to support the resolution of problems of the real world. Solving problems implies making choices, among some existing alternatives, on the way the problem should be solved. The choices that a decision maker picks are unarguably crucial, as they can determine the failure of the project; maybe choices cannot always determine the success of the project, which can depend also on external and uncontrollable factors, but we can be fairly sure that bad choices are likely to lead to bad failures.

The model we want to build should help make such choices, according to the set business goals and constraints. We name this model the “Impact Vector Model”.

Several attempts of supporting decision makers, at different levels, exist in the history of Software Engineer, e.g.: CoCoMo for estimating software cost and supporting resource allocation; GQM approach for measuring software development and support team managers understand where to improve it; architectural, design and implementation patterns to establish a set of reusable solutions to help developers build good software. These achievements share the idea of making the world of software development more understandable, manageable, engineered and predictable, while keeping the flexibility and tailorable that software development requires.

The step further we propose in this work consists in formalizing the concept of choice in our thinking, be it a technological choice (e.g. a cutting-edge technology rather than a well-proven technology), a process choice (e.g. performing a code inspection rather than using pair programming) or a project choice (e.g. acquiring a COTS and a consultant rather than developing in-house).

We do not expect to finalize the construction of a general model, to apply in all possible contexts and for all possible choices. We rather expect to propose a way of organizing, representing, accessing and leveraging knowledge gathered experimentally in the course of decades via research and on-the-field experiences. This way, our model could be instantiated in each single context, where it will be leveraging context-specific data and will propose solutions tailored down to the context itself.
Also, our model is thought to be suitable for extension and refinement via long-term, multifaceted and collaborative work, where experienced professionals and far-looking scientists can join their efforts to leverage, augment and improve it, without breaking what has already been put in place, possibly exploiting and sharing the same data and repositories.

Chapter IV. Conceptual Architecture of the Model

So far, we have presented many good intentions, but nothing really practical has existed, yet. Section II will fix this issue, but first readers may like to have a high-level understanding of how this model is expected to work. Therefore, a conceptual view of the model is provided in this chapter, identifying inputs, outputs, actors and the general idea of the model functionality.

Consider a typical industrial context, where the decision maker is the one responsible for a new project that the company has to lead off. Management sets goals and constraints for the project and, based on some contextual factors, the decision maker has to make choices. Such choices, according to constraints and goals, are made based on the decision maker’s experience and knowledge and are transformed into tasks for the development teams. The model we propose can be leveraged during the decision process to support the decision step, i.e. help the decision maker formulate tasks for the development team. Other than goals and constraints set by management, the model will probably take some additional information, e.g. project-specific or team-specific characteristics or needs, additional constraints on the team composition or task assignment, or technical information and data to be used by the model for qualitative or quantitative reference.

The desirable output of the model should some reliable information the decision maker can immediately leverage in order to elaborate or refine choices and tasks for the development teams. The term “reliable” means that the output of the model should be based on some evidence, i.e. qualitative and quantitative information with some form of objectivity and confidence on the part of the decision maker and the top management.

Some form of iteration is also introduced: choices may reveal some team behaviors or results which differ from the expectations and may require further actions; similarly, the model could suggest paths that require an interaction with top management (e.g. the acquisition of new skills or products), which may lead to the adjustment of budget or time constraints, as well as to the introduction of new constraints.
Of course, the main question is: how can we build such a smart, evidence-based model? First of all, we want the model to be able to leverage mathematical and statistical analysis, as well as data mining techniques. This implies the existence of data, or, more precisely, a data-driven approach enabling such methods to operate, with the aim of producing some “reliable supportive information” for making evidence-based decisions.

Readers willing to have a concrete example will need to wait until the end of the mathematical formalization of the model. Our next step, in fact, is the definition of a mathematical foundation, which will unveil part of the internal structure of our black box.
Section II: Impact Vector Model

«There are problems to whose solution I would attach an infinitely greater importance than to those of mathematics, for example touching ethics, or our relation to God, or concerning our destiny and our future; but their solution lies wholly beyond us and completely outside the province of science».

Carl Friedrich Gauss
Chapter V.  Shaping the Model

V.i. Assumptions

The model that will be presented founds on some assumptions that need to be discussed and made explicit in order to agree and find soundness in the rest of the work.

The first point is to clarify the term “model”, which we will use pretty much freely, sometimes as synonym of theory – i.e. a working hypothesis that is considered probable based on experimental evidence or factual or conceptual analysis and is accepted as a basis for experimentation – and sometimes to indicate the mathematical formulation that will be provided in the next Section. Eventually, model, theory and mathematical formulation will correspond to the same idea (from a Platonic point of view), which is the same represented in Figure 1. We name this the “Impact Vector Model” (IVM). Reasons for such name will be clear later on.

The second point to focus is that the IVM requires the awareness of its users, i.e. the decision makers. This means that the model cannot replace the human judgment, but can only assist it and support it during the decision making process. This is required to compensate all the missing aspects of the reality, i.e. those aspects which are not accounted for by IVM, including global economic trends, future opportunities, personal opinion, political influences, hard-to-model dynamics, and, very frequently, gaps in available data.

Since we do not aim to model “the whole”, nor we do expect this being feasible, the underlying assumptions are:

(A1) We accept to deal with an incomplete-knowledge model.

(A2) We assume the presence of a user capable to interact with the model and interpret its output accounting for contextual factors that are not modeled explicitly.

The first assumption is pretty much true for every model of the real world, including physical and mechanical laws. The counterpart of this assumption is that the model is required to be “complete enough” to provide usable and useful information. The achievement of “enough completeness” will be shown in the rest of the work.

The second assumption is hopefully true in many safety- and mission-critical environments, including the Space and Defense Fields, where decision makers are expert, prepared and skilled enough to address both technical and managerial problems. In particular, we expect the decision makers to have a broad view on both the system under development and its context; this means that the user should also be able to figure out when the model cannot be applied, e.g. because of the high incompleteness of data or the singularity of the specific issue to solve.
Based on this second assumption, we derive that the field of applicability of the model is limited to those contexts in which A2 holds.

V.ii. Characteristics of the Model

Given these assumptions, we want the IVM to be used in the following scenario:

Given:

(i) Some product and process goals and constraints (e.g. a given quality or a maximum project time);
(ii) The current status of both process and product;
(iii) The history of the process executed so far;
(iv) Potential solutions coming from standards, literature, practice and creativity;
(v) Data collected overtime (on the current and other projects);
(vi) Information on the specific context;

the model should provide a set of possible process solutions that will lead to achieving the project goals while respecting the given constraints.

This scenario includes, as particular case, the situation in which the decision maker is at the beginning of the development and no history exists.

An important characteristic we want the IVM to have is that it should be evidence-based: a particular instance of the IVM should be the result of a systematic empirical investigation of phenomena. This means that the model is built on the basis of qualitative and quantitative data. Furthermore, its output should be measurable as well, i.e. supported by measures associated to a well-defined interpretation.

One more characteristic is the real-time capability: we want the model to constantly provide the decision maker with fresh and reliable information, which can be used to monitor the progress and to make decision on how to correct the plan or improve it.

Finally, very desirable characteristics of the model also include:

- **Flexibility**: we want the decision maker to be able to define his/her own goals on any characteristic of the product, the process or the context.
- **Usability**: we want the model to provide actionable and quickly understandable information.

V.iii. Taxonomy

We start formalizing the model by providing details on some terms we will use. Most of them are widely spread in the field of SE, but conclusive definitions are always hard to provide and
agree upon. We do not want to speculate on what definition is most appropriate for any given term, so we will not provide any formal definitions at all, but just a few words to transfer an intuition of our interpretation of the terms.

- **Development process**: a multi-set of process activities executed according to a given schedule.
- **Development process activity**: a task that requires resources to be executed and that causes a development process to evolve because of its execution.
- **Product**: an artifact or set of artifacts, either tangible or not, which is output of some development process activities.
- **Product attribute**: a measurable characteristic of a product.
- **Process attribute**: a measurable characteristic of a process.
- **Project attribute**: a measurable characteristic of a project.
- **Context attribute**: a measurable characteristic of the environment where a process is enacted to realize a product.
- **Scale**: a model which enables a variable to be measured and assigned a value in some set of admissible values.

Chapter VI. **Mathematical Foundation**

VI.i. **Formulation**

We now define our model by providing some formal definitions and properties.

1. Be $X \subseteq \mathcal{X} = \{x| x$ is a product attribute, process attribute, project attribute or context attribute$\}$. $X$ is the subset of $\mathcal{X}$ containing only the attributes we are interested in considering.
   Every element in $X^k$, for any given $k \geq 0$ is named **Impact Vector** (IV).

2. Be $A \subseteq X^n = \{a| a$ is a development process activity$\}$, where $n = |X|$ and $X^n$ denotes the $n$-fold Cartesian product over $X$. Furthermore, an activity is defined as a vector of product attributes, process attributes, project attributes, and context attributes with no repetitions, i.e.:

«Do not worry about your difficulties in Mathematics. I can assure you mine are still greater».  
Albert Einstein
∀a ∈ X^n. a = \vec{x} = \langle x_i \rangle$ and $i \neq j \rightarrow x_i \neq x_j$  \hspace{1cm} (F. 1)

We name such vector \textbf{Activity Performance Impact Vector (APIV)}.

3. Be $A \subseteq X^n$ the set of all software development activities; then, a development process is a sequence of zero or more activities, i.e., be $D$ the set of all development processes, then:

$D \subseteq (X^n)^*$ \hspace{1cm} (F. 2)

We name \textbf{Process Performance Impact Vector (PPIV)} an element in $(X^n)^*$.

4. Be $V = \{v | v$ is a scale$\}$.

5. Be the \textit{admission function} $adm: X \rightarrow V$, i.e. the function that maps each attribute to the set of its admissible values (i.e. to its scale).

   o It holds that each attribute has one and only one scale, i.e.:

$\forall x \in X \exists! \ v \in V : v = adm(x)$ \hspace{1cm} (F. 3)

6. Be $\text{Dist} = \{\text{dist} | \text{dist is a probability distribution function or a mass function on } v \in V \}$ the set of all probability distribution/mass functions on any scale.

7. The \textbf{Activity Performance Function (APF)} $\text{perf}: A \ast D \rightarrow (X \ast \text{Dist})^n$ represents the vector of $n$ probability density/mass functions, reminding that $n = |X|$. Informally, we are saying that, given an activity $a$ in a process $d$, the performance of the activity is a vector of $n$ probability distribution/mass functions, each representing the performance in terms of an attribute.

8. The \textbf{Combination Function (CF)} $\text{comb}: (X \ast \text{Dist})^n \ast (X \ast \text{Dist})^n \rightarrow (X \ast \text{Dist})^n$ maps a pair of APIVs of a process - each associated to its performance - to a single Impact Vector, having its attribute associated to their performance (i.e. random variables). Informally, we are reducing the performance of two activities in a process to the performance of one equivalent activity.

9. The \textbf{Process Performance Function (PPF)} $\text{ppf}': D \rightarrow (X \ast \text{Dist})^n$ for a process $d$ of $k$ activities is defined recursively as follow:

$\text{ppf}'(d_j) = \begin{cases} \text{comb}(\text{ppf}'(d_{j-1}), \text{perf}(a_j,d)) & \text{if } 0 < j \leq k \\ \langle \text{null} \rangle & \text{if } j = 0 \end{cases}$ \hspace{1cm} (F. 4)

where the notation $d_j$ indicates the sub-development process composed of the first $j$ activities $(0 \leq j \leq k)$ of the process $d$ and $\langle \text{null} \rangle$ denotes the \textbf{Null Impact Vector (NIV)}, whose properties we will define later.

Something to highlight is that, based on this definition, the performance of each activity (and, in particular, the performance of each activity in terms of each attribute) depends on the entire
process; in fact, the second parameter of $comb$ is $perf(a_j, d)$, which computes the performance function for the activity $a$ as $j$-th activity of the process $d$. We now want to make an assumption to simplify the model:

**A3** The performance of an activity does not depend on the future of the development process.

Therefore, the definition of the PPF $ppf$ for a process $d$ composed of $k$ activities becomes:

$$ppf(d_j) = \begin{cases} 
comb(ppf(d_{j-1}), perf(a_j, d_{j-1})) & \text{if } 0 < j \leq k \\
\text{null} & \text{if } j = 0
\end{cases}$$

In this final formulation of PPF, the performance of the $j$-th activity depends on the process up to the $(j-1)$-th activity, not on the ones which will be executed after the $j$-th activity.

It is important to remark that the choice of an activity can obviously depend on the future of the process; nevertheless, the independence from the future, that we are assuming, is limited to the performance of that activity, not to the absence of correlations or synergies among activities during process planning. However, the assumption A3 might still be violated, e.g. because some psychological dynamics can improve or worsen the performance of some developers who are aware of next planned activities. Some studies report quantitative analyses on these dynamics – e.g. “An Empirical Study on Software Engineers Motivational Factors”, by França and da Silva –, but, in order to keep the complexity of this model at a manageable level, we will keep assumption A3 until the end of this work and demand the problem of managing the social and motivational factors to the decision maker’s consideration (see assumption A2).

One more important point to notice is that the model does not forbid that the performance of an activity in terms of an attribute may depend on the performance of other attributes. In fact, this reflects something realistic: just to give an example, we can imagine that there should be attributes like “complexity of the product” and that the expected cost of testing depends on (i.e. is a function of) such complexity.

Notice also that, at the moment, the APF (Activity Performance Function) $perf$ and the CF (Combination Function) $comb$ are still undefined and we postpone their discussion to the next chapter.

**VI.ii. Using the model**

We now provide an example of use of the model. Given the trivial development process $d_1 = <$requirements; implementation; test$, we want to compute its performance in terms of cost, recall (i.e. percentage of removed defects) and amount of produced material (apm).

According to the previous paragraph, we can write:
• \( X = \{ \text{cost, recall, apm} \} \)
• \( D = \{d_1\} \)
• \( A = \{ \text{requirements, implementation, test} \} \)

Moreover, we arbitrarily define the following scales:
• \( V = \{ V_{\text{cost}}, V_{\text{recall}}, V_{\text{apm}} \} \)
  o \( V_{\text{cost}} = \{ c | c \text{ is a non-negative real number} \} \).
    ▪ The real number is assumed to represent the cost in $. 
  o \( V_{\text{recall}} = \{ \text{not applicable, low, high} \} \).
    ▪ The value “not applicable” means that an activity having this value in the recall dimension does not affect the recall anyhow. The value low indicates that less than 33% of existing defects are detected by the activity, while high means that at least 33% of the existing defects are identified by the activity. 
  o \( V_{\text{apm}} = \{ \text{very small, small, medium, high, very high} \} \).
    ▪ The values of the ordinal scale indicates if the amount of pages is very small (0-2 pages), small (3-5), medium (6-10), high (11-25), or very high (>25). We can think of some conversion factors, e.g. for code, transforming lines of code to pages. However, in this example, we are not really interested in this level of detail and we prefer to leave the intuition to the reader.

Additionally, the admission function is completely defined as follow:
• \( \text{adm} = \{ <\text{cost}; V_{\text{cost}}>, <\text{recall}; V_{\text{recall}}>, <\text{api}; V_{\text{apm}}> \} \)

We can also arbitrarily assume that the performance function is defined as follow:
• \( \text{perf}(\text{requirements},<>)=P_{\text{requirements}, <>} \)
• \( \text{perf}(\text{implementation},<\text{requirements}>)=P_{\text{implementation},<\text{requirements}>} \)
• \( \text{perf}(\text{test},<\text{requirements, implementation}>)=P_{\text{test},<\text{requirements, implementation}>} \)

where \( P_{*,<*}> \) are vectors of 3 random variables – one each for cost, recall and apm – associated to the probability distribution/mass functions constituting the co-domain of the function \( \text{perf} \) (i.e. the APF). Let us assume we somehow know that such random variables have the following distribution and mass functions:
• \( P_{\text{requirements}, <>} = < \)
  
  <cost, Normal(5000,2)>
  
  <recall, < <100\%, \text{not applicable} > > >
Now we need to compute \( \text{ppf}(d_1) \) and the steps are provided below.

In order to support the reader, the two arguments of the functions \( \text{comb} \) and \( \text{perf} \) are formatted and numbered hierarchically, according to their depth in the function call stack. Also, we highlight with colors some invocations; the same color marks the same invocation of the \( \text{comb} \) and \( \text{perf} \) functions at different steps of the computation.

\[
\text{ppf}(d_1) = \\
= \text{ppf}(<\text{requirements}; \text{implementation}; \text{test}>) = \\
= \text{comb} ( \\
\quad 1. \text{ppf}(<\text{requirements}; \text{implementation}>), \\
\quad 2. \text{perf} ( \\
\quad\quad 2.1. \text{test}, \\
\quad\quad 2.2. <\text{requirements}, \text{implementation}>) = \\
= \text{comb} ( \\
\quad\quad 1. \text{comb} ( \\
\quad\quad\quad 1.1. \text{ppf}(<\text{requirements}>), \\
\quad\quad\quad 1.2. \text{perf} ( \\
\quad\quad\quad\quad 1.2.1 \text{test}, \\
\quad\quad\quad\quad 1.2.2 <\text{requirements}, \text{implementation}>) ), \\
\quad\quad 2. \text{perf} ( \\
\quad\quad\quad 2.1. \text{implementation}, \\
\quad\quad\quad 2.2. <\text{requirements}>) = \\
= \text{comb} ( \\
\quad\quad 1. \text{comb} ( \\
\quad\quad\quad 1.1. \text{comb} ( \\
\quad\quad\quad\quad 1.1.1 \text{ppf}(<\text{requirements})) \\
\quad\quad\quad\quad 1.1.2 \text{perf} ( \\
\quad\quad\quad\quad\quad 1.1.2.1 \text{test}, \\
\quad\quad\quad\quad\quad 1.1.2.2 <\text{requirements}, \text{implementation}))), \\
\quad 2. \text{perf} ( \\
\quad\quad 2.1. \text{implementation}, \\
\quad\quad 2.2. <\text{requirements}>) = \\
= \text{comb} ( \\
\quad\quad 1. \text{comb} ( \\
\quad\quad\quad 1.1. \text{comb} ( \\
\quad\quad\quad\quad 1.1.1 \text{ppf}(<\text{requirements})))) \\
\quad 2. \text{perf} ( \\
\quad\quad\quad 2.1. \text{implementation}, \\
\quad\quad\quad 2.2. <\text{requirements}>) = \\
= \text{comb} ( \\
\quad\quad\quad 1. \text{comb} ( \\
\quad\quad\quad\quad 1.1. \text{comb} ( \\
\quad\quad\quad\quad\quad 1.1.1 \text{ppf}(<\text{requirements})) \\
\quad\quad\quad\quad\quad 1.1.2 \text{perf} ( \\
\quad\quad\quad\quad\quad\quad 1.1.2.1 \text{test}, \\
\quad\quad\quad\quad\quad\quad 1.1.2.2 <\text{requirements}, \text{implementation}))), \\
\quad 2. \text{perf} ( \\
\quad\quad\quad 2.1. \text{implementation}, \\
\quad\quad\quad 2.2. <\text{requirements}>) = \\
= \text{comb} ( \\
\quad\quad\quad\quad 1. \text{comb} ( \\
\quad\quad\quad\quad\quad 1.1. \text{comb} ( \\
\quad\quad\quad\quad\quad\quad 1.1.1 \text{ppf}(<\text{requirements})) \\
\quad\quad\quad\quad\quad\quad\quad 1.1.2 \text{perf} ( \\
\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2.1 \text{test}, \\
\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2.2 <\text{requirements}, \text{implementation})))), \\
\quad 2. \text{perf} ( \\
\quad\quad\quad 2.1. \text{implementation}, \\
\quad\quad\quad 2.2. <\text{requirements}>) = \\
= \text{comb} ( \\
\quad\quad\quad\quad\quad 1. \text{comb} ( \\
\quad\quad\quad\quad\quad\quad 1.1. \text{comb} ( \\
\quad\quad\quad\quad\quad\quad\quad 1.1.1 \text{ppf}(<\text{requirements})) \\
\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2 \text{perf} ( \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2.1 \text{test}, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2.2 <\text{requirements}, \text{implementation})))), \\
\quad 2. \text{perf} ( \\
\quad\quad\quad 2.1. \text{implementation}, \\
\quad\quad\quad 2.2. <\text{requirements}>) = \\
= \text{comb} ( \\
\quad\quad\quad\quad\quad\quad 1. \text{comb} ( \\
\quad\quad\quad\quad\quad\quad\quad 1.1. \text{comb} ( \\
\quad\quad\quad\quad\quad\quad\quad\quad 1.1.1 \text{ppf}(<\text{requirements})) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2 \text{perf} ( \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2.1 \text{test}, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad 1.1.2.2 <\text{requirements}, \text{implementation})))), \\
\quad 2. \text{perf} ( \\
\quad\quad\quad 2.1. \text{implementation}, \\
\quad\quad\quad 2.2. <\text{requirements}>) = \\
\)
1.1.1. **ppf** (<>)

1.1.2. **perf** ( 

1.1.2.1 test, 

1.1.2.2 <requirements, implementation>)

1.2. **perf** ( 

1.2.1 implementation, 

1.2.2 <requirements>)

2. **perf** ( 

2.1. requirements, 

2.2. <>))

We do not need to expand the expression any longer, in fact: **ppf**(<>) is known and is the Null Impact Vector (NIV) by definition of **ppf**; the **perf** function is known and explicitly defined for all required values of its arguments; and the combination function **comb** is expected to provide an element in $X^3$, even though it has not been defined, yet. We can intuitively say that, for cost, the **comb** function is a simple sum: computing the cost of two activities is equal to add the cost of the first activity to the cost of the second activity (computed being aware of the application of the first activity). More specifically:

$$Normal(5000,2) + Normal(1500,1) + Normal(1000,0.1) \sim Normal(7500,3.1)$$

For recall and apm, instead, we have a different scale type (ordinal) and we cannot have an immediate similar intuition (and the same would hold for nominal scales). Concerning the recall, however, it is still fairly easy; in fact, only one of the activities has a relevant value, i.e. test, because the “not applicable” value of the scale means that an activity does not affect the recall. Therefore we can say that the recall of the process is a random variable with a mass function of the form <<50%, low>, <50%, high>>.

Chapter VII. **Addressing the Open Problems of the Impact Vector Model**

In the defined models a number of issues can be identified. In the following, the most relevant ones are pointed out and a solution is sketched for each.
VII.i. Accounting for Parallel Process Activities

As stated in the previous chapters, the current mathematical formulation of the Impact Vector Model does not support parallelism of process activities, i.e. multiple activities being executed at the same time.

In order to model the parallelism of activities, we extend the definition of activity to include a new attribute. This attribute is a pair representing an interval of execution of the activity with respect to a nominal time (i.e. a reference time). When two activities $a_1$ and $a_2$ are executed at the same time, e.g. they start at the same time $t_1$ and end at the same time $t_2$, the activity attributes have both the pair $<t_1, t_2>$. When the time overlap between the two activities is partial, then the nominal time reflects this partial overlap; e.g. if $a_2$ starts when half $a_1$ has been executed, then the attribute value for $a_1$ is $<t_1, t_2>$ and the one for $a_2$ is $<t_{1.5}, t_3>$. Nominal time shall be measured in a ratio scale.

Following this extension, the nominal duration of the process is $t_f - t_i$, where $t_f$ is the end time of the last activity of the process (i.e. the one ending last), while $t_i$ is the start time of the first activity of the process.

A further extension can support modeling activities that are not continuous, but that can be suspended and resumed in the future; the extension consists in having a sequence of pairs instead of a single pair as attribute. Pairs of this sequence shall be totally ordered with respect to time; therefore there should be no overlap between intervals of execution (i.e. an activity cannot be ongoing and suspended at the same time).

The introduction of such an extension (either the one supporting activity suspend-and-resume or the former one) has an impact on the complexity of the problem: in fact, without the extension, interactions and synergies between activities only depend on their order of execution, but not on their simultaneity. By introducing the concept of time, instead, simultaneity of activities may lead to much higher a complexity: multiple activities may interact and provoke changes in performance and causality can become hard to understand. E.g. a new question may arise: which activity (or activities) is causing a performance variation on which other activity (or activities)? An exponential grows of potential answers shall be expected with the increase of the number of simultaneous activities. Furthermore, also partial overlap of activities shall be accounted for.

Because of this complexity, an exhaustive analysis of all possible and reasonable situations is not viable. However, extending the Impact Vector Model to support simultaneous activities is necessary, as time is of undeniable importance in every practical context. Nevertheless, despite the uncountable realistic scenarios one can encounter, it is expectable that in many contexts only some activities are executed simultaneously, as well as only some sequences of activities.
are implemented, while most admissible combinations, though reasonable, are not of interest. This can cut down the complexity of the theoretical problem to a more practical and treatable task. This way, decision makers can perform targeted investigations, e.g. if they detect that a particular activity is sometimes executed parallel with another activity (e.g. they start and ends at the same time), while sometimes they are executed in isolation, they could investigate the existence of an overall performance variation by targeting those two specific process configurations (i.e. parallel or isolated activities); assuming this standpoint avoids an extensive and exhaustive empirical investigation trying to cover all possible configuration of process activities.

VII.ii. Impact Vector Constraints

One more aspect we want to report is that, in some conditions, not all values of a scale are reasonable or feasible (e.g. a project manager wants to put a cap to costs, which makes some values of the corresponding scale not acceptable). How can the model account for such conditions?

In order to satisfy this need, we formally introduce the concept of Attribute Constraint (AC), i.e. a function that, given a scale of values, maps it into a new scale of values that is a subset of the former scale. Even though the formalization effort for this extension is small, it turns into great benefits in practice. In fact, putting ACs may filter out a big set of alternative solutions that, for a given context, are not feasible, not viable or considered not convenient.

The usefulness of ACs become evident when trying to build automation engines to elaborate impact vectors: by putting constraints, the search space can be drastically reduced, making the computation accomplishable in acceptable time.

One more type of constraints relates to activity sequencing: some activities may be executed only after some other activities have been already executed, e.g. because they require artifacts produced by some previous activities. For similar situations, we formalize sequencing constraints by extending the Process Performance Function (PPF) as follow:

$$ppf(d_j) = \begin{cases} 
  \text{comb}(ppf(d_{j-1}), \text{perf}(a_j, d_{j-1})) & \text{if } 0 < j \leq k \\
  \text{null} & \text{if } j = 0 \\
  \text{undefined} & \text{if } a_j \text{ must be executed before and activity of } d_{j-1}
\end{cases}$$

The undefined value means that the performance of the process $d_j$ cannot be computed. Such formulation further helps create engines for automatically elaborating impact vectors, because the search space can be reduced by excluding undefined sequences of activities.
VII.iii. Empirical Definition of the Activity Performance Function and Combination Function

The next point we need to cope with concerns the $perf$ and $comb$ functions. That is probably the hardest aspect to define, but it leads to the most valuable result of using the Impact Vector Model, i.e. the process performance, both expected and historical, at the level of granularity chosen by the decision maker.

For executing a performing development process, some good practices can often be identified and leveraged, as well as some bad practices can be detected and discouraged. However, on the side of these practices, in order to i) find local, tailored and specialized optima, ii) concretely and practically instantiate such practices in specific situations, and iii) interpret the obtained results, every context needs its own models.

Therefore, other than globally valid information (e.g. best practices), there is some local information to account for. Such local information is specific to a given context, but it is not necessarily one-shot. In a totally different environment, this local knowledge might still be useful. Let us analyze when this could happen.

When configuring the Impact Vector Model for a given context, one of the steps is the selection of performance attributes of interest. Among such attributes, big importance has to be assigned to characterization attributes, i.e. those attributes that represent distinctive aspects of the process and the product under development, e.g. the complexity of the project and the domain of application. Doing so corresponds to preparing statistical models and data mining agents for a correct analysis, because such attributes allow slicing and dicing of data in order to find meaningful trends.

For example, suppose a company C1 is working on two product lines, one about mobile GUIs for existing websites and one about native mobile apps for home automation; an aggregate analysis can reveal some global and context-specific knowledge, but a by-product line analysis may reveal some different and product line-specific knowledge. Packaging such product line-specific knowledge can represent an added value to be leveraged by someone else, when the product line is appropriately characterized. Keeping on this example, in fact, we can imagine of a Defense company C2 that has to develop a wrapper for its intranet portal to offer functionality on mobile. Data collected by C2 are hardly usable to predict the performance on such task; but if C2 could access data of C1, limited to the product line on mobile interfaces, then C2 could gather something useful, e.g., C2 could get to know that the presence of one person highly skilled on the target platform may cut down cost by 40%, in the average, and up to 60% when the target platform is iOs. The aforementioned appropriate characterization of the context could help C2 find the needed information via data look up and exploration.
We postpone to the last Section of this document a discussion on the so called Empirical Software Engineering 2.0 [1], which matches our viewpoint on sharing data across companies and researchers.

Jumping back to the point of this chapter, we can now imagine of having a repository with data as previously described, i.e. vectors of attributes representing the performance of an activity and including characterization attributes to use for data look up and exploration. Such a repository could be queried and automatically analyzed via statistics and data mining to predict what the performance of an activity (or an entire process) will be, based on the characterization of the current context to slice and dice the repository and use only the data representative and really usable for a prediction in that specific situations, in terms of both a single activity performance (Activity Performance Function), and multiple activities (Combination Function).
Page intentionally left blank.
Section III: Impact Vector Model: Leveraging the Excellence

«Not to know what happened before you were born is to remain forever a child».

Marcus Tullius Cicero
In Section III we perform a comparative analysis between the Impact Vector Model and some previous work in the field of Software Engineering, specifically for process and quality Management during software development.

Chapter VIII.  Where do Impact Vectors Come from?

The Impact Vector Model is the results of leveraging, reinterpreting, merging, slicing and dicing, adapting and augmenting a number of pre-existing models in software – and not only – literature. The biggest influence has been provided by the Quality Improvement Paradigm, by GQM+S, and by the Value-Based approaches applied to Software Engineering. For each of those, a chapter provides a summary description and a close examination on how they interact and relate to the Impact Vector Model. A minor, but still crucial contribution to our model has been given by further influent paradigms, namely CMMI, SPICE and Six Sigma. Additional methods and techniques should obviously be leveraged to implement the Impact Vector Model, e.g. the Weighted Objectives Model, Statistical Process Control, Root Cause Analysis, Regression Analysis, checklists and so on. In the next chapters, we will mostly focus on the aspects that the Impact Vector Model has in common with other approaches and methodologies, and sometimes we will mention techniques or methods that can be used to address specific technical issues. However, we do not aim to carry on a detailed comparative analysis between impact vectors and other approaches, but only to highlight what impact vectors are leveraging and reusing from the past.

Chapter IX.  Enabling Value-Based Decisions via the Impact Vector Model

The Value-Based Software Engineering (VBSE) is a discipline of Software Engineering that heavily accounts for business aspects related to software development. Among the pioneers of VBSE is Barry Boehm. One of the books he co-authored – “Value-Based Software Engineering” – contains, in its first chapter, a number of key-drivers for the definition and use of the Impact Vector Model. The re-elaboration and synthesis of Boehm’s words, reported below, highlights the most relevant aspects for our model.

“Most studies of the critical success factors distinguishing successful from failed software projects find that the primary critical success factors lie in the value domain. In earlier times, when software decisions had relatively minor influences on a system’s cost, schedule, and value, a value neutral approach was reasonably workable: every requirement, use case,
object, test case and defect was treated as equally important and the responsibility of software engineers was confined to turning software requirements into verified code.”

However, “software has a major impact on cost, schedule and value” of today’s systems. Furthermore, value-neutrality clashes with the definition of engineering itself, which recalls to the application of science and mathematics in order to produce results useful to people.

The Value-Based Software Engineering (VBSE) aims at “integrating value considerations into the full range of existing and emerging software engineering principles and practices”. This enables making value-based decision, i.e. decisions driven by the awareness of what is most valuable doing in order to achieve the defined business goals. In particular, decision makers “need to have a clear understanding of the sources of value to be exploited and link between technical decisions and the capture of value”.

“Value-based decisions are of the essence in product and process design, the structure and dynamic management of larger programs, the distribution of programs in a portfolio of strategic initiatives, and national software policy. Better decision making is the key enabler of greater value added”.

These words remark the substantial need of including value considerations when trying to address decision making issues. In the cited book, the authors also shape a theory of value for SE, elaborating, adapting and leveraging ideas and theories of value defined in fields other than SE. That is only one of the works that have been done on VBSE and we can reasonably assert that a fairly strong and fervid background exists for VBSE. Though interesting, we will not focus on the details of VBSE, as it is not the core of this work. Rather, the goal of this paragraph is to focus how the Impact Vector Model enables value-based decision making. In order to focus this

![Figure 2 Geometrical representation of the Impact Vector Model.](image)

3

Measurement-Based Decision-Making
point, we want to propose a graphical representation of the Impact Vector Model. Because of the limits of our paper sheets and our ability to deal with 2 dimensions only, we will propose a trivial example in which we consider only two attributes (A1 and A2) and three processes (D1, D2 and D3).

In Figure 2 many elements are represented. D1, D2 and D3 are represented as sequences of, respectively, 4, 1 and 5 activities, where each activity is indicated as a bold dot. All development processes start from the origin, i.e. the Null Impact Vector. Every activity is a move in the 2D space we are considering. The move corresponds to the impact that the activity has on the development process. Each point in the space (i.e. an Impact Vector), in fact, represents a performance in terms of A1 and A2. In particular, each point of type \(a1\), \(a2\) indicates that the performance is \(a1\) in terms of A1 and \(a2\) in terms of A2. The big green dot represents an optimum, i.e. the point that, ideally, the decision maker would like to achieve. For example, if we think of A1 and A2 as time and cost, the green dot represents the ideal tradeoff for a process in terms of cost effectiveness, i.e. a time \(o1\) and a cost \(o2\). Finally a portion of the space is marked with a curved line, which represents a limit to the admitted values for the attribute A2, i.e. an Impact Vector Constraint. Again, if A2 represents cost, then the value \(c\) for A2 is the maximum budget that can be used by a process we want to consider as acceptable.

In this example, no development process is expected to get to the ideal performance (green dot). We have three options, but one of them is to be discarded. In fact, the process D2 ends up in the forbidden space, as it violates the constraints on A2. D1 and D3 are the remaining candidates: how can the decision maker choose? Here comes the idea of value-based decision making. Based on the graphical representation, the decision maker should start reasoning on the relative importance of A1 and A2. In fact, if we still think of A1 and A2 as time and cost, D3 seems to cost less than D1, but it is expected to take longer. Which alternative is the best? The answer definitely depends on the context. If a new project is expected to start shortly, the decision maker may want to be faster to complete the current project, even though it is more expensive, so to free resources for the new projects. In different conditions, however, the decision can be different; therefore the model cannot really provide the conclusive answer. What the model can do is to provide the decision maker with the entire amount of information available in order to him/her to make a value-based decision.

We also want to remind that another desirable characteristic is the evidence-based foundation of the model: how can we introduce quantitative reasoning to support value-based decisions, then? A perfect fit could be the Weighted Objectives Method (WOM), a quantitative method described by Cross in his “Engineering design methods: strategies for product design”, and commonly used in the field of Systems Engineering to evaluate a set of candidate solutions to the same technical problem. It consists in computing a single score for each alternative and
picking the solution corresponding to the highest score. In our context, the alternatives are the
development processes (e.g. D1 and D3), while the scores can be computed by following four
steps:

1. Assign a weight to each dimension of the space (e.g. cost and time), so that the higher
the weight, the more important the dimension; notice that the weights add up to 1.
2. Define a ratio scale, with minimum and maximum values, representing the extent to
which a given alternative is appropriate with respect to the given dimension.
3. For each dimension, assign a value of the scale associated to the current alternative;
notice that, regardless of the dimension, the method works in such a way that the
higher the value, the better is the alternative (e.g., in terms of cost, the cheapest
solution is expected to have the highest score).
4. For each alternative, sum the scores assigned to each dimension multiplied by its
weight.

Notice that, for dimensions originally associated to a ratio scale, step 2 can be skipped and step
3 is systematic, i.e. the score for those dimensions can be analytically computed based on the
value presented by each alternative for those dimensions. For ordinal and nominal scales,
instead, step 2 and 3 are the enablers to the use of the weighted objectives method and must
be executed.

WOM is just one of the possible methods; it is easy to apply and does not require a lot of effort.
On the other side, there are some more accurate methods that have been proposed in the past
[2, 3, 4, 5].

However, the selection of the most appropriate method, which may also be context-specific
and vary according to the available resources and support tool, is not the focus of this chapter.

The point that we want to make in this chapter is that with any of such methods, we can give
rigor to the way the decision maker could leverage the information represented geometrically
in Figure 2, in order to support his/her decision making in a value-based fashion.

![Figure 3 An example of impact vectors](image-url)
We also want to remind the importance of flexibility, listed in 0 as one of the desirable characteristics of the framework. Being able to accommodate different needs (i.e. different goals) is a first step towards flexibility and, by using the WOM together with impact vectors, as different weight assignments to vector attributes can lead to different outcomes and consequently to different choices. Of course, using WOM is not mandatory and a decision maker can pick any preferred method to plug onto the impact vector to assign importance to the attributes.

Chapter X. QIP and GQM: Continuous Process Improvement via Impact Vectors

X.i. QIP: Quality Improvement Paradigm

The Quality Improvement Paradigm (QIP) is a model designed to support continuous process improvement and engineering of the development processes, and to help in technology infusion. The QIP takes the premise that all project environments and products are different; in fact, software development is human-based and design-intensive, which makes the use of statistical control (e.g. used for manufacturing) extremely hard to apply. Therefore, the idea of statistical control of processes is brought into question and replaced, at least partially, with the need for constant experimentation.

The QIP closed-loop approach is represented in Figure 4. The QIP cycle is composed of two closed loop cycles: organizational and project. The organization-level cycle provides feedback to the organization after the completion of a project, whilst the project-level cycle provides feedback to the project during the execution phase, with the goal of preventing and solving problems, monitoring and supporting the project, and aligning processes with goals.

Additional details on each phase of

![Figure 4. Quality Improvement Paradigm.](image-url)
QIP will be provided later on, when describing the GQM, which can leverage, and is complementary to the QIP (and vice versa). During that description, we will also remark how the Impact Vector Model complies with and extends QIP (other than GQM).

X.ii. GQM+S: Goal, Question, Metric + Strategies

The Goal, Question, Metric (GQM) is an approach to software measurement proposed by V. R. Basili and the Software Engineering Institute at the NASA Goddard Space Flight Center. The approach is based on three concepts: goals, questions, and metrics.

Goals can be defined according to the GQM template as follows:

<table>
<thead>
<tr>
<th>Schema</th>
<th>Examples</th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Process, product, resource...</td>
<td>Analyze &lt;...&gt;</td>
</tr>
<tr>
<td>Purpose</td>
<td>Characterize, understand, evaluate, predict, improve</td>
<td>For the purpose of &lt;...&gt;</td>
</tr>
<tr>
<td>Quality Focus</td>
<td>Cost, correctness, reliability...</td>
<td>With respect to &lt;...&gt;</td>
</tr>
<tr>
<td>Viewpoint</td>
<td>Customer, manager, developer, corporation...</td>
<td>Form, the point of view of &lt;...&gt;</td>
</tr>
<tr>
<td>Context</td>
<td>Problem factors, resource factors, process factors...</td>
<td>In the context of &lt;...&gt;</td>
</tr>
</tbody>
</table>

Table 1. GQM template.

Starting from defined goals, questions shall be generated, so to make the goals quantifiable. Such questions should address those aspects relevant for the achievement of the goals, e.g., they could aim at identifying relevant process areas and at determining the influential characteristics of the product, and context factor that impact the performance of teams.

Given the questions, measures shall be specified to collect data in order to answer the generated questions and track process/product progress with respect to the goals.

A concept underlying GQM is that measurement is not a goal per se, but it is functional to an end which has a value for the organization. Such value is expressed in the form of goals. The extension of GQM to GQM + Strategies (GQM+S) addresses specifically this issue by introducing...
the concept of strategy, i.e. a possible approach for the achievement of a goal in the context under study. Consequently, a multi-level hierarchy of goals and questions should be created in order to sufficiently address, decompose and detail the top-level goals, so to accommodate organizational structure and tractability of the problem.

In order to precisely illustrate how the Impact Vector Model can be plugged in to GQM+S (or vice versa, depending on the point of view), we will first be traveling through the process to implement the GQM+S approach in an organization; then, we will explain which steps of the process can be affected and augmented by the Impact Vector Model.

The process can be summarized in 7 steps, which retrace the QIP (steps 2-7).

1. Initialize. The management should be made aware of and prepared to implementing the approach. This includes staffing, assigning responsibilities, training, and planning schedule and cost.

2. Characterize. Processes, products, projects, technologies, customers and organization should be characterized, as well as pre-existing measurement programs (e.g. repositories, goals or models).

3. Set goals. Organizational units (e.g. business units) involved should be identified and interrelationships should be detected. Furthermore, the current status should be precisely described and goals should be set for each organizational unit, along with potential issues, strategies for achieving the goals and data to be used for assessing the goals. This step includes the definition of both metrics and interpretation models to evaluate success at achieving goal.

4. Choose process. Create or reuse measurement programs to collect and analyze data, providing the appropriate level of detail to precisely specify who is responsible for collecting which metrics and the technical procedure to collect them (schedule and technologies).

5. Execute model. Run the strategies, collect and monitor data, assess consistency and completeness and provide real time feedback and adjustments.

6. Analyze results. Filter and prepare data, create descriptive statistics, visualize, and explain and evaluate results according to the interpretation models and baselines.
7. Package and improve. Disseminate results, perform a cost-benefit analysis and store reusable and relevant outcomes. Connect to step 2 for next round of goals.

![GQM process diagram]

X.iii. Modeling Choices and Alternatives with Impact Vectors

The Impact Vector Model settles on and augments the steps of QIP and GQM. First of all, we focus on the process, which is mostly shared by QIP and GQM.

After the first step of initialization, the GQM proceeds with the characterization phase; in term of Impact Vector Model, during step 2, the set X of “process, product, project and context attributes” shall be defined. From a geometrical point of view, characterizing the elements listed in step 2, in fact, consists in defining the Impact Vector space, i.e. the dimensions of the impact vectors.

During step 3, the objective Process Performance Impact Vector (PPIV) is defined. Multiple business goals can still be summarized in a single PPIV: its coordinates, in fact, encompass all interesting attributes and different objective values can be set on any number of attributes.
Step 4, i.e. the definition and planning of data to collect, and methods for their collection, is conceptually unchanged and it consists in the determination of actions to collect data and organize them in the form of impact vectors.

During step 5, the geometrical interpretation of the Impact Vector Model provides a quantitative way of measuring and evaluating the progress. The distance between the three points - initial PPIV, current PPIV and objective PPIV – represents the extent to which the goal has already been achieved, i.e. its progression. The projection of the current PPIV onto the dimensions on which goals have been set during step 3 represents the progress with respect to each single goal. Based on such progress, the decision maker could enact some changes. However, GQM+S does not provide the decision maker with any guidelines or methods on how to pick the best alternative, because actually the concept of “alternative strategy” is not explicit in GQM+S.

This does not mean that GQM+S is ignoring this issue: step 7, in fact, includes storing the data during last iteration into a knowledge base. Such knowledge is something that can be leveraged to make decisions, and, in step 4, it is explicitly stated that existing measurement plans should be accounted for and reused as possible to set new goals and make choices.

Nevertheless, no systematic ways to reusing previous plans are part of GQM+S. More in general, the concept of alternative is not explicitly dealt with in the approach and choices are demanded to implementers’ craft. Here the Impact Vector Model can provide its contribution. When previous and potential choices are modeled as impact vectors, estimations and predictions can be built upon them, each of which leads to a point in the impact vector space. In practice, instead of choosing a single strategy, a decision maker picks the entire set of known possible alternatives and builds an augmented GQM graph. This way, something similar to Figure 3 can be created. The next action has already been described in 0: value-based decision making is enabled and the decision makers can apply the preferred technique to quantitatively come up with the most promising strategies. These strategies will be the nodes in the new GQM graph and the GQM+S process will continue.

As already pointed out, the Impact Vector Model can be plugged in also to step 5 of the process, i.e. when real-time feedback are provided at project-level instead of corporate-level; in this case, the type of choices will be different, e.g. selecting a technology or a testing tool, but the procedure for determining the most promising strategies remains the same.

It is fairly obvious, but still useful to remark, the high influence of QIP on GQM+S, and, in turn, of QIP and GQM+S on the Impact Vector Model. In particular, impact vectors perform best only in presence of a continuous learning cycle and when evidence-based approaches are planned to pervade the software development process. Furthermore, since the organization of the impact
vector repository, as well as the use of impact vectors, requires some form of analysis, synthesis and re-elaboration, it would be useful to have an ad hoc group mediating development teams and repository, e.g. an Experience Factory [6, 7, 8], so to minimize the formalization effort and the population of the space of alternatives.

Chapter XI. Impact Vector Model: Points of Contact with CMMI, ISO/IEC 15504 and Six Sigma

**Capability Maturity Model Integration (CMMI)** [9]: this is a process improvement training and certification program and service administered and marketed by Carnegie Mellon University. CMMI aims to increase the business value produced by the organization via enhancement of the development process, and our Impact Vector Model shares this CMMI aim.

According to CMMI authors, CMMI can be used in process improvement activities as a collection of best practices, i.e. a framework for organizing, prioritizing and coordinating activities. However, CMMI is not a process, but rather it describes the characteristics for an effective process to have. CMMI holds both for improving technical processes, including development processes, service processes, and acquisition processes.

In order to obtain “process improvement”, CMMI identifies 22 “process areas”, so that activities composing a process to improve can be mapped to one or more process areas to address. For each process area, there are specific “goals”, for which specific “practices” are expected. A process area is a cluster of related practices that, when implemented collectively, contribute satisfy a set of (CMMI) “goals” considered important for making improvement in that area. Depending on which practices are being implemented, an organization can be rated from level 2 to level 5, which corresponds to the capability of the organization. In order to achieve a given level (capability), a pre-determined subset of process areas must be addressed; every level adds some process areas to the ones that are necessary to achieve the lower level.

Specific practices can be composed of “sub-practices”, i.e., detailed descriptions providing guidance for interpreting and implementing a specific practice; in our taxonomy, the CMMI terms “specific practice” or “sub-practice” correspond to “activity”. The CMMI idea of “goal”, in the Impact Vector Model, is meant only in terms of measurement, i.e. our requirement is that each CMMI goal should be stated by defining a goal vector, i.e. a PPIV. Therefore, the definition of a goal in our model is more stringent (more similar to “goals” in the GQM paradigm) than CMMI goal definition.
Along with “specific practices” and “specific goals”, as implicitly referred above, CMMI identifies “generic practices” and “generic goals”. These lasts describe the characteristics that must be present to institutionalize the practices that are implemented for a process area. Generic practices are activities that ensure that the practices associated with the process area will be effective, repeatable and lasting; they are named generic because they appear in multiple process areas. For our purpose and from a formalization standpoint, the distinction between specific and general practices and goals is marginally important. Conceptually, each application of an activity, including the ones referable to generic practices, is an instance of impact vector. Impact vector attributes (i.e. the coordinates of each impact vector) do not have an equivalent in CMMI, which postpones measurement to maturity levels 4 and 5 and delegates to process managers the identification of suitable metrics and their management, interpretation and use.

<table>
<thead>
<tr>
<th>Level</th>
<th>Focus</th>
<th>Process Areas</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Optimizing</td>
<td>Continuous process improvement</td>
<td>Organizational Innovation &amp; Deployment</td>
<td>Productivity &amp; Quality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Causal Analysis and Resolution</td>
<td></td>
</tr>
<tr>
<td>4 Quantitatively Managed</td>
<td>Quantitative management</td>
<td>Organizational Process Performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Quantitative Project Management</td>
<td></td>
</tr>
<tr>
<td>3 Defined</td>
<td>Process standardization</td>
<td>Requirements Development</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Technical Solution</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product Integration</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Verification</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Validation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Organizational Process Focus</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Organizational Process Definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Organizational Training</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Integrated Project Management</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Risk Management</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision Analysis and Resolution</td>
<td></td>
</tr>
<tr>
<td>2 Managed</td>
<td>Basic project management</td>
<td>Requirements Management</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project Planning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project Monitoring &amp; Control</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supplier Agreement Management</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Measurement and Analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process &amp; Product Quality Assurance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configuration Management</td>
<td></td>
</tr>
<tr>
<td>1 Initial</td>
<td>Competent people and heros</td>
<td>Processes &amp; Products</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8 CMMI levels and capabilities

Summarizing, concerning companies interested in achieving CMMI levels, impact vectors may play a useful role as soon as an organization aims to achieve levels 4 or 5: use of collected metrics for process improvement are effectively and efficiently viable by using the impact vectors; more specifically, the main targets of the Impact Vector Model are the following process areas:
• “Quantitative Project/Work Management” (L4),
• “Organizational Process Performance” (L4), and
• “Organizational Performance Management” (L5).

During maturity levels lower than 4, impact vectors can still be leveraged for the following process areas:
• “Measurement and Analysis” (L2),
• Project/Work Monitoring and Control (L2),
• Project/Work Planning (L2),
• Process and Product Quality Assurance (L2),
• Integrated Project/Work Management (L3),
• Organizational Process Definition (L3), and
• Decision Analysis and Resolution (L3).

At CMMI L2 and L3, the Impact Vector Model can provide qualitative support and a structured approach of addressing decision problems; at L4 and L5, instead, with data being collected systematically, decisions should be based on quantitative information and impact vectors can be totally exploited for control and enhancement purpose. In fact, they can be combined to produce prediction models for different performance attributes, including quality attributes, cost for activities and sub-processes, and every attribute required to perform a quantitative software development management.

ISO/IEC 15504 information technology – Process assessment [10], also known as Software Process Improvement and Capability Determination (SPICE), is a set of technical standards documents for software development process. In SPICE, processes are organized in five process “categories”: customer/supplier, engineering, supporting, management, and organization. Each process is assigned a “capability level” between 0 and 5; the level is measured using nine “process attributes”, each consisting of one or more “generic practices”. Then, each generic practice is composed of “practice indicators”, used to assess performance. Performance “assessment” utilizes the degree of fulfillment of process attributes, as measured in the scale {Not-achieved, Partially-achieved, Largely-achieved, Fully-achieved}; values of this scale are assigned to attributes based upon evidence collected against the practice indicators. In other words, while the CMMI appraisal is standardized, the SPICE assessment is not entirely standardized; in fact, the conversion from an indicator to the degree of fulfillment is standardized, but the translation of collected evidence into an indicator is not unambiguously defined. However, some form of tailoring and flexibility is accepted (and actually planned) for both evaluation processes.
Similarly to the comparison with CMMI, also for the case of SPICE impact vectors can represent a good fit for supporting process management and improvement since the beginning of the path (level 1); however, only at the highest capability levels they become most useful, when measurement, control and predictability become keywords for further process enhancement.

**Six Sigma** is a set of tools and techniques for process improvement originally developed by Motorola in 1981 for its manufacturing and business processes. One of the assumptions of Six Sigma is that the development process has “characteristics” that can be measured, analyzed, controlled, and improved. Based on this assumption, Six Sigma encourages continuous efforts to enact a predictable development process, which has to be measured, and includes the definition of quantified value targets, e.g. process cycle time reduction, increase of customer satisfaction, or cost reduction. Value targets correspond to what we call PPIV, hence GQM goals, and the term “characteristics” to be measured is equivalent to our “attributes”. Also in Six
Sigma, as well as in the Impact Vector Model, identifying characteristics that are “critical to quality” (the so called CQs) is one of the most important steps to perform either to identify or to improve, depending on the methodology being (DMADV for process definition or DMAIC for process improvement). Additionally, in the case of definition of a new process (or, by extension, the introduction of new sub-process in an existing process), it requires the identification of alternatives to implement the process, which is something we have already focused on in the previous chapter on GQM+S, when we introduced the concept of choice among different strategies to augment the GQM+S methodology.
Page intentionally left blank.
Section IV: Impact Vector Framework

«In almost everything, experience is more valuable than percept».

Marcus Tullius Cicero
In Section IV, we present a tool chain that supports the use of the Impact Vector Model in practical contexts. The first tool of the tool chain – Impact Vector Schema - is a piece of Java software of our creation, while the rest of the tool chain leverages existing tools currently used for different purposes.

Chapter XII. Impact Vector Schema: Organize and Store Impact Vectors

For the realization of a tool for organizing, storing, and managing impact vectors, we took inspirations from a paper by Sira Vegas and Victor Basili: “A Characterization Schema for Software Testing Technique”, which focuses on software verification technique characterization. According to such paper, a verification technique can be characterized by a hierarchy of levels, elements and attributes. Each technique is characterized across several levels, each level is composed of several elements and each element is composed of several attributes. Each attribute is assigned a value of some data type.

The term “attribute” in the aforementioned paper has a meaning alike the one we use in the Impact Vector Model; in fact, if we admit that the value of an attribute can be a probability distribution function on a scale, the term “attribute” makes sense both in the V&V schema by Vegas and Basili, and in the Impact Vector Model. However, the V&V Schema admits free text values for its attributes; even though the Impact Vector Model does not explicitly forbid the use of free text for some attributes, it is recommended that admissible values be machine-understandable, so to enable automatic analysis. Consequently, the Impact Vector Model requires a higher degree of formalism in the technique characterization than the V&V schema.

Also, V&V schema expects to have one sheet (i.e. one instance of the schema) for each verification technique. In the case of the Impact Vector Model, instead, we expect to have a number of impact vectors referring to the same techniques, i.e., an impact vector for each execution of that technique. Subsequently, we can combine such impact vectors referring to the same technique and build a new impact vector that represents the expected performance of that technique, so to help predict the performance of such technique in new applications. In the V&V schema, knowledge about the activity is aggregated in a single sheet (e.g., the cost of a technique falls approximately in a given range, which is supposed to be true for every execution of that technique), which is good for humans, but not precise enough for the automatic processing purpose we defined nor for fitting the Impact Vector Model approach.

In conclusion, in order to realize a tool to store impact vector effectively, we can take inspiration from the V&V schema, but we need to place additional constraints, i.e. the ones
defined in the model formalization of the Impact Vector Model. Furthermore, we need to widen the focus to including techniques other than verification. Also, structuring attributes in “elements” and “levels” may represent a good idea to cope with management issues: when the number of attributes becomes hardly treatable by a single human, having them organized hierarchically could be an effective way of focusing the attention on those dimensions most relevant to the specific context. In order to allow such structure, however, we do not need to add further formalism in the Impact Vector Model: we intend elements as sets of attributes and levels as sets of elements, but nor elements nor levels do offer specific properties or do have a peculiar theoretical meaning; they are just a way of aggregating attributes in an easier-to-manage hierarchy.

One last remark is that one could expect a first identification of some relevant attributes with their consequent organization in levels and elements. However, we recall that the importance of an attribute strictly depends on the context of application. This means that an attribute which is important in a particular context may have no interest in a different context; for example, the “all-p-use coverage” metric of an automatic test case generation tool may be relevant for a safety-critical project, where such a coverage metric may be mandatory to collect and maximize up to a given value, but it may have no interest for a web entertainment application project. Therefore, we do not want to propose here a set of attributes and their structure; we rather postpone such a task to Section V, where we will report on some practical applications of the Impact Vector Model and will show some attributes we identified.

In the following of this chapter, we list the main requirements we elicited and implemented for IV-schema, i.e. the tool we realized to store and structure the impact vectors.

**XII.i. Impact Vector Schema: Main Requirements**

Requirements are organized in two sets: the first set includes requirements for managing the impact vectors (data), while the second set includes requirements for managing the structure of the data (dimensions and their hierarchical organization). The following synthesis lists some main requirements implemented for the beta version.

A. **Managing the structure of the data**
   a. **Manage attributes**
      i. **List attributes**
         1. By techniques giving it a value: not all impact vectors have a valid value for each attribute, as some attributes make no sense for some impact vectors (e.g. all-p-uses coverage for a risk management activity)
   b. **Manage activities**
c. Manage scales
d. Manage elements
e. Manage levels

B. Managing the impact vectors
   a. List
      i. By technique
      ii. By attribute
      iii. By level
   b. Create and associate to a technique
c. Edit
   i. For a single impact vector, i.e. those attributes not shared by other impact vector because referred to a specific instance of an activity (e.g. efficiency attributes).
   ii. For all impact vectors referring to the same activity (e.g. name, element or description)
d. Delete impact vector
e. Export to datasheet

“Manage” requirements include create, read, update and delete operations, as well as export functions.

XII.ii. Impact Vector Schema: Architecture and Technology

Typical Software Engineering best practices were employed in the realization of the tool, along with well-known technologies and frameworks. Just to mention some of them: a tailored version of IBM RUP as development process (iterative and incremental); Java as development language; JBoss Seam as development framework, configured to work with JSF and RichFaces, Javascript and Ajax for the view layer (other than HTML+CSS), and Hibernate for the data layer.

A view of the data model is reported in the following E-R diagram; the RDBMS used for the current version is MySql, but a migration to DB2 is planned for reasons which will be made explicit later in this work. For an easy understanding, a “record” in the model represents an impact vector instance, while a “v vnode” represents a technique. “Value” and “datatype”, instead, are the way we currently represent a scale, i.e. by means of a member (value) and its semantics (datatype); for this beta version, this representation is enough, but in the future it will require refactoring and improvement in order to accommodate the implementation of algebras for attributes.

The rest of the entities and relationships is intuitive and matches our taxonomy.
The business logic is implemented via the Home Controller pattern, heavily used by JBoss Seam; the business logic model is reported in the following UML class diagram.
The application server used to test the tool is JBoss AS and the integrated development environment is Eclipse. The complete and exact technology stack is as follow:

- MySQL Community Server. v5.1
- MySQL Workbench. v5.2
- MySQL Java Connector v5.1
- Eclipse IDE for Java EE Developers. v.3.6 Helios SR2
- JBoss Tool. v3.2
- JBoss Application Server. v.6.1.0.Final
- JBoss Seam. v2.2.2.Final (including Hibernate)
- Apache Ant. v1.8.2-4

The current version of the tool can be found on [www.mastrofini.it/iv-schema](http://www.mastrofini.it/iv-schema).

Chapter XIII. **Using IBM Cognos Insight™ to Visualize, Explore, and Analyze Impact Vectors**

Once impact vectors and their structure are exported as Excel files, they can be imported in more sophisticated tools for visualization, exploration and analysis. In particular, we identified IBM Cognos Insight™ [12] as an adequate tool for managing multidimensional data, doing reporting and facilitating visualization and exploration of impact vectors.

Cognos Insight is a tool for Business Intelligence (BI), where by BI we mean a set of theories, methodologies, processes, and technologies to transform raw data to information useful for business performance improvement. Cognos Insight permits to study business performance, identify patterns and trends in the data, and plan improvement strategies. Among its main concepts, we highlight the following: (i) **workspace**: a composition of graphical elements (widgets) for the analysis, exploration, filtering, and visualization of business data; (ii) **dimension**: set of pieces of information that describe an aspect of the business, e.g., geographic dimension or time dimension; (iii) **level**: a slice of a dimension, e.g. region, nation, and city are three levels of the geographic dimension; (iv) **measure**: a quantifiable performance indicator, e.g., sell price or margin; (v) **cube**: a multidimensional representation of business data like dimension and measure.

Cognos Insight can import data from csv, xls, xlsx, txt, asc, tab, cma file formats, but also from proprietary formats and from IBM databases. For this reason, we plan to port the IV-Schema tool from MySQL to DB2, which is natively supported by Cognos Insight. When importing data, Cognos Insight allows to map data to match the planned structure with its inner representation.
In particular, Cognos Insight tries to automatically map data to identify hierarchies and measures. This is an important feature: as we have seen in the previous chapter, in fact, we want to group and organize impact vectors in hierarchies (i.e. level, elements and attributes). With the only issue of taxonomy mismatches, we can easily map IV-Schema (or, equivalently, V&V Schema) members to IBM Cognos Insight members. What Cognos Insight calls “measure” is actually a dependent variable, i.e., a performance indicator that has to be optimized in order to increase the business value produced by the company. In the Impact Vector Model, every attribute can play the role of Cognos Insight measure, but, of course, some of them make more sense, depending on the context and on the needs. Therefore, during the mapping step, a decision maker can indicate which attributes being imported should be represented as measures in Cognos Insight.

A limitation of Cognos Insight is that it can manage non nominal scales of measure. However, nominal scales can be dealt with as non-measures and obtain an equivalent effect in terms of visualization, exploration and analysis capability. One more limitation, instead, is that the data types “probability distribution function” and “mass function” are not managed. On the other side, visualizing this type of data may be problematic and we are currently investigating ways to cope with this issue, as well as whether resolving this issue can bring some added value to the tool chain and to the entire framework.

Additional features of the tool include: drill up and drill down; sorting; exploration points (for filtering and organizing data); calculations; time rollup. Cognos Insight offers a number of chart types, that can be automatically generated starting from the available data. Slicing and dicing of data is really efficient when done via Cognos Insight and it reveals a number of aspects of interest of the data under study.

Finally, commercial versions of Cognos Insight also allow to share workspaces and strategies among users, so to enable spreading the information across the company and a collaborative approach to data production and provisioning, according to the established access rights to the platform.

However, Cognos Insight only supports qualitative analyses and reporting, so additional tools are needed for quantitative statistics, hypothesis testing and data mining. IBM SPSS Modeler can be integrated with Cognos Insight, as the first can read and write data for the latter. SPSS supports the creation of predictive analytics and advanced statistical analysis, so to complement most of the functionality required for data analysis. On the side of IBM tools, there are many other offerings, both commercial (e.g. by Microsoft and Oracle) and free/open source (e.g. Weka). For time constraints, we did not yet perform an extensive comparative study to identify the statistical and data mining tools that best fit the structure of our model.
However, once we jump in the statistical or data mining world, the concept of impact vector becomes less visible and has a minor influence on the way such analyses are performed; therefore, we do not need not investigate this aspect any longer in this work and redirect the interested reader to more focused books and manuals.

Chapter XIV. IBM Method Composer™ & IBM Team Concert™: Infuse, Estimate, Manage, Measure, and Improve with Impact Vectors

One last point of interest, on which we want to focus for the definition of a tool chain for impact vectors, is on how to concretely employ the Impact Vector Model via existing tools used to manage the development process.

The first tool we want to focus on is IBM Rational Method Composer (RMC), a flexible process management platform, built on top of Eclipse, with a method authoring tool and a process asset library to help implement measured improvement.

Through RMC, one can create, tailor, manage, and publish process descriptions. In particular, among the many features, we want to highlight the possibility of creating estimation models. As explained in the IBM information center (http://pic.dhe.ibm.com/infocenter/rmchelp/v7r5m1), an estimating model is a way of calculating estimates that can use any number of estimating factors. An estimating factor is something that has a quantitative influence on the effort required to perform a task or activity. Examples of estimating factors are use case complexity (complex, medium, simple), business areas, platforms, screens, classes, use cases number. In order to compute an estimate, estimating formulas are used. The estimating formula is a relatively simple algebraic equation and the estimating factors are the parameters used in that equation. After the estimating factors are defined, the estimation formula can be used to calculate the estimated effort required to complete a task or work breakdown element. The result of the calculation is typically expressed in hours.

Estimating factors can be associated with a task or activity, and estimates and summary values can be computed via a predefined list of formulas that cannot be edited. As well as for the Impact Vector Model, RMC provides a bottom-up estimating approach. Estimates are created for tasks in a work breakdown structure (WBS) for a project and summed through the WBS to arrive at an estimate for the whole project. The depth where the bottom up approach is selected is not necessary a leaf of the WBS, but one can select any level and then climb the WBS structure up to the root of the project.
Summarizing, RMC fixes the set of combination functions to a predefined one, allows to estimate the effort of an activity or task, and supports the definition of estimation factors (i.e., impact vector attributes). Even though the combination function is a crucial part of the Impact Vector Model and the effort is not the one dependent variable we are interested in estimating, RMC resulted as the most useful tool to infuse the Impact Vector Model in the practice of an enterprise. Its native integration with Rational Team Concert (RTC), another IBM tool for lifecycle management, allows carrying the impact vectors throughout the process tasks in teams’ everyday work that can be characterized, planned, monitored, measured, and improved with a fine-grained development process quantitative management.

For further details on RMC and RTC, we demand the reader to the next section, where examples of application and integration between these tools and the Impact Vector Model are proposed.

Again, we want to remark that our survey of the state of the practice was not exhaustive, but rather driven by our industrial partners. For the future, we plan to investigate new tools and alternatives to support the Impact Vector Model.

In conclusion, as of today, the tool chain suggested for using the Impact Vector Model in a practical context is represented in Figure 13.

![Figure 13. The recommended tool chain end-to-end for using the Impact Vector Model.](image-url)
Page intentionally left blank.
Section V: Impact Vector Model: Experimentation
In this Section, excerpts from our published papers and our work executed at companies’ are reported, as documentation of the practical application of the Impact Vector Model.

Even though we did not have the time to fully employ impact vectors in a single, encompassing context, we actually had the chance to experiment nearly all parts of the framework in isolation and in different contexts, with different goals and different issues. Therefore, for presentation reasons, we made the decision to create an abstract company, the Impact-One, and report our real experiences as they were performed in this context.

Furthermore, given the criticality and sensitiveness of the content of our studies, we will often obfuscate the results and provide only partial views, according to the restriction and classification policies enacted by our partners. So, the reported results, though representative and realistic, do not necessarily reflect the reality.

We catch the opportunity to mention the organizations that contributed to the experimental validation of our model and supported our investigations. In order of intervention during the validation process:

- University of Rome Tor Vergata, Italy (2010-2013);
- MBDA Italia, Italy (2011-2012);
- Fraunhofer Center for Experimental Software Engineering - Maryland, USA (2012-2013);
- NASA, USA (2012);
- IBM Italia, Italy (2012);
- Keymind, USA (2013).

Chapter XV. **Impact-One: Virtuous Company Leveraging the Impact Vector Framework**

_XV.i. First Steps for Applying the Impact Vector Model: Context_

The Impact-One is a philanthropic, young, innovative and florid organization operating in the field of complex medical devices development. The devices developed by the company operate in an integrated way, so to form a network of devices collecting a lot of data from many patients. Data are stored on some servers geographically distributed that interact and exchange data transparently to the devices. Then, Impact-One technology supports physicians for their diagnoses via a distributed web application, which assures privacy to patients while interactively guiding the physician through the evaluation of plausible alternatives via data exploration, including data collected on other servers.
Therefore, Impact-One IT progress requires a diverse and broad set of skills within the software division.

The company has so far outsourced the development of all pieces of software to a couple of companies; however, the software development, though not under direct control of Impact-One, has always been executed in tight connection and continuous collaboration with the rest of the company teams. As of now, management has decided to incorporate software development into the company as an internal, new division, with the goal of reducing software development costs while maintaining the current quality of produced software. In order to achieve this goal successfully, the Quality Division has been assigned a new software quality assurance head: Jed Squa. Squa’s goal is to support the company at defining a sustainable, predictable, innovative, and high-end software development process; he is highly skilled in the impact vector model methodology.

Without entering the organizational details of all the Impact-One’s divisions, we just want to list them and provide some overview information:

- Development division (cyan squares and yellow squares on the left side of Figure 14), including the 3 groups of Electronics, Mechanics, and System (cyan), now extended to 4 groups with the inclusion of Software (yellow);
- Quality division (green and brown squares, in the middle of Figure 14), with one Quality Assurance (QA) group per development group (green), including software QA (brown);
- Human Resources division (dark blue squares on the right);
- Marketing division (very dark blue squares on the right);
- Finance division (light blue squares);
- Administration division (very light blue squares), with its Legal and Accounting groups.

If we focus on the Software Development group within the Development Division, the following teams exist: Architecture and Analysis (3 people); Design (4 people); Coding (11 people). Software QA (SQA) group within the Quality Division is composed of 8 people: a SQA lead (i.e., Jed Squa), a team leader for each used software verification technique (i.e., test, formal methods, simulation, and inspections), a responsible for software reliability, one more for software safety and the last one for software process. SQA team leaders have the authority to require that Development people be assigned to assurance activities. Software Development group and SQA group form the so called software intergroup.

Since many consultants have been working full time at the company for several months, some of them were acquired as employees and already have a good baseline experience with the domain; however, a few new people were also hired and assigned to the Software Development group.
All development groups, but software, have a well-established development process, clear standards to follow, and interaction dynamics among groups and teams are consolidated. The part of those dynamics that is most interesting to the new software group is the interaction with the system group, which takes care of: (i) system requirements definition, breakdown and allocation to subsystems; (ii) system integration for putting together the developed subsystems, assess and possibly forward change requests coming from other development groups. Therefore, the software team is expected to integrate with those dynamics and get to be part
of the game. This lets Squa believe that the most critical point be project time: a not well consolidated team, whose interaction dynamics are not established nor known, may lead stretching development time unexpectedly. For this reason, particular care to project time control shall be taken. On the other side, it would be unreasonably expensive to implement the same development process both for the embedded software and for the web platform. Therefore, a very rigorous process shall be proposed for the embedded software, while a more flexible and free process shall be implemented by the teams working on the web software.

XV.ii. First Steps: Activity Identification, Definition, and Characterization

Once given the general description of the company, we focus on the software intergroup. More specifically, we want to begin by focusing on the work of establishing a development process for the embedded software; this step starts by identifying and defining process activities required to be performed. As part of activity definition, also their characterization in terms of measurable indicators is expected. In the taxonomy of the Impact Vector Model, Dr. Jed Squa well knows that this means to define the set of attributes we are interested in considering in the context under study. This is the first, mandatory step to start applying the Impact Vector framework.

After a deep analysis of the state of practice and the state of the art, under suggestion of Dr. Squa, the company proposed to adopt the IBM Rational Harmony Process for Embedded Software™ [13], tailored to accommodate the organizational structure, contexts, needs, and goals. Aside this process, the IBM Rational Rhapsody™ tool was selected as Integrated Development Environment. Afterwards, a picky analysis of each single activity and task in Harmony has started, along with a review of the related literature with the goal of concretely and precisely characterize the activities’ performance in a measurable way.

As an example, if we take the task “Definition of the architecture”, the identified metrics of interest are:

- Average time spent to define each architectural element;
- Average time spent to define all communication links for a given architectural element;
- Average time required to generate all architectural elements related to a single use case.

It should seem obvious that those metrics are some sort of dependent variables, because we would like to tune the activity – i.e., to manipulate some other attributes of the activity – to optimize those metrics (i.e., to reduce the average times as much as possible without compromising the quality). Attributes to manipulate include:
• Presence in the architecture definition team of an architect with more than 20 years of experience, a Boolean value;
• Presence in the architecture definition team of an architect with at least 10 years of experience, a Boolean value;
• Presence in the architecture definition team of an architect with at least 5 years of experience, a Boolean value;
• Number of people directly involved in the definition of the architecture, an Integer value;
• Average number of years of experience of all involved architects, a Real value;
• Degree of innovation of the selected technologies (e.g. frameworks or tools), a value in an ordinal scale: very low (no unused technology), low (1 new technology selected, for which a well historical record of application exists in literature), medium (2-5 new technologies selected, for which well historical records of application exist in literature), high (1 new technology selected for which no historical record of application exists in literature), critical (more than one new technology selected for which no historical records of application exist in literature)
• Average number of requirements change requests generated per architecture element.

So, the first three metrics, along with the last 6 metrics, are dimensions of the space in which the impact vectors will live and evolve.

Every task of our tailored version of Harmony will provide its contribution to the Impact Vector Space by possibly introducing further dimensions. So the number of dimensions can increase significantly. However, not all dimensions are relevant to every task of Harmony: this can be managed by assigning to the insignificant dimensions their zero value. E.g., the average number of requirements change requests generated per architecture element could have an irrelevant influence on the performance of tasks like unit testing (actually, it does not even make any sense for such a task). Based on the formulation of the Impact Vector Model, Dr. Squa knows that this fact does not represent a big deal: the analysis of the data collected overtime will elicit this fact when building the Activity Performance Function (we will later see how that happens).

In this paragraph, we have given a partial example on the type of activity characterization. In the next paragraph, we provide a much more detailed description of the proposed process, including not only activities, but also tools, roles, and built-in metrics that can help address our main goal: project time control. We also show how all descriptions were inserted in IBM Rational Method Composer™, the tool we mentioned and recommended for use with Impact Vectors. Nevertheless, we do not focus on the independent variables, i.e. those characteristics
of the activities that need to be tuned in order to achieve the planned goal. This further aspect (i.e. the independent variables) is the object of the next Section.

Chapter XVI. Tailoring the IBM Rational Harmony for Embedded Software

In this part of the work, we elaborate on the IBM Rational Harmony process for the development of embedded devices, as the ones being developed by Impact-One.

XVI.i. Process Description

As previously mentioned, Systems Engineers provide the other development groups, including the Software Group, with some subsystem requirements specification. Starting from these requirements, the Software Group is expected to execute the following 20 activities, which we group by 7 phases.

I. System Requirements Analysis
   A1. Analyze Stakeholder requests;
   A2. Define System validation plan;
   A3. Specify Requirements and trace them (e.g., Traceability matrix) to Stakeholder requests;

II. System Functional Analysis
   A5. Identify and trace Use cases to Requirements (Context Diagram, CD, and Use Case Diagram, UCD).
   A7. Define and trace System test plan.
   A8. Verify Black box use case design through execution and produce related Reports.

III. Design Synthesis
   A10. Define System architecture (Internal Block Diagram, IBD).
   A11. Define White box use case design (AD+SD+I&P+SM+BDD+IBD).
A13. Verify White box use case design through execution and produce related Reports.

IV. **Unit Implementation and Test**
A15. Develop or generate units and trace them to White box use case design.
A16. Define and trace Unit test plan.
A17. Perform unit test, identify bug resolution strategies, and correct problems.

V. **System Integration Test**
A18. Perform system integration test, identify bug resolution strategies, and correct problems.

VI. **System Test**

VII. **System Validation**
A20. Perform system validation.

Figure 15 is a graphical representation of the process, reporting phases, activities, generated artifacts, allowed paths, and decision gates.

The process is incremental, in the sense that the system can be decomposed into subsystems (or parts) which may be developed independently, either by different teams, at different time, or in different increments, depending on schedule and priorities. Obviously, for each decomposition step, a corresponding integration step shall be planned.

The process is completely iterative. Specifically, it is within-phase iterative and between-phase iterative.

It is within-phase iterative because each one of the phases is iterative. Activities within that phase may be executed as many times as needed in order to achieve the required level of detail and fix all encountered problems, without enacting any particular policy. E.g., when specifying requirements (activity A3, Phase “System Requirements Analysis”), it is possible to come back to the system validation plan and update it.

The process is between-phase iterative because it is possible to go back to previous phases as many times as needed. In fact, between each couple of phases, there is a decision gate. Given a phase $n$, at its end, the sunny day path (green outgoing arrow from the decision gate) allows to move to the next phase $n+1$, whereas the rainy day path (red outgoing arrow from the decision gate) takes back to the previous decision gate, i.e. the one between phase $n-1$ and phase $n$. Such a decision gate chain creates the possibility to rollback to previous phases of the process.
However, developers cannot freely go back from any activity of a given phase to an activity of a different (previous) phase. In order to go back to an activity of a previous phase, in fact, developers need to go through the decision gate at the end of the phase they are currently in. This is not a limitation, because the process is incremental and each one phase is iterative, thus there is no risk to fall back to a waterfall process model. Furthermore, it is obvious to notice that, when an issue is detected, developers may make some steps back up to the last passed decision gate.

The presence of decision gates, which are the unique points to switch from one phase to another, helps enforce the process. Suppose that, for example, during activity A11, the team detects a defect, e.g. a missing use case. The team goes back to decision gate G3, where the following question shall be answered: Is the detected defect caused by “missing or incorrect black box use case design?” In case it is, the team goes back again, up to decision gate G2. Suppose that no requirement is missing, thus activity A5 “Identify use case and trace to requirements” is performed, where the identification of the new use case is followed by its tracing back to requirements. If a free iteration between activities of different phases were allowed, then developers could have directly performed A6 and possibly skipped some fundamental activities, i.e. traceability. Enforcing the developers through decision gates aims to avoid such kind of situations and keep the process consistent.

To pass through the milestones in the following enables the team to proceed to the next phase. For each iteration, a milestone needs to be planned and met, in order to pass through the decision gate. One more milestone (the first) is also planned after the stakeholder requests analysis.

M1. Understood Stakeholder requests.
M2. Reviewed and approved Requirements, Traceability, System validation plan, and Risk analysis document.
M3. Reviewed and approved Black box use case design, System test plan, Reports, and Risk analysis document.
M4. Reviewed and approved Architecture, White box use case design, System integration test plan, Reports, and Risk analysis.
M5. Implemented units, and performed and passed unit test.
M6. Integrated units, and performed and passed system integration test.
M7. Integrated all units, and performed and passed system test.
M8. Performed and passed system validation.

As part of the Quality Improvement Paradigm, i.e. the approach leveraged to define the proposed process, also a set of metrics for goal-driven process measurement were defined: metrics aim to provide quantitative data to assess the time required by of each process activity.
This should, in turn, provide information to project leaders to control the process and to pursue the goal of reducing project time. The proposed metrics are the first step for the definition of a measurement strategy, which planned to be consolidated during the next iteration of the proposed methodology.

**XVI.i.1. Process Description: an Example**

Figure 16 shows an example of the application of our tailored Harmony process. Specifically, the example is composed of two nested executions of the process: the outer execution is at system-level, the inner at software level. Process phases are named accordingly to the Harmony process, i.e., Requirements Analysis, Functional Analysis, and Design Synthesis. At the end of the execution of the last phase at system level, unit implementation should be planned. As explained in paragraph “Develop or Generate Units and Trace to White Box Use Case Design”, such activity may encompass a further application of the process at a lower level of details. Specifically, it consists in the development of a software subsystem. Software subsystem development passes through the same phases as system development. This time, the unit implementation (i.e. software implementation) consists in generating the source code.

Verification phases and artifacts, as well as non-functional requirements management, are not reported for presentation reasons. In short, the process execution is:

- **System Requirements Analysis**: given the Stakeholder requirements, System requirements are derived.
- **System Functional Analysis**: based on System requirements, the System use cases group System functional requirements at Black box design level, and SysML diagrams specify system functions.
- **System Design Synthesis**: for each System use case, a set of SysML System blocks is created and detailed, each representing a part of the system. This way, System white box design is created.
- **Software Requirements Analysis**: a subset of System requirements is allocated to software, and the Software Requirements Engineering discipline derives Software requirements from System requirements.
- **Software Functional Analysis and Design Synthesis**: based on Software requirements, the Software use cases group Software functional requirements. Each Software use case shall also derive from a System use case and implement it in *toto* or part. This way, Software black box design is created.
- For each Software use case, a set of UML Classes is created and specified, each deriving or being part of at least one SysML System Block. This way, Software white box design is created.
• Given UML Classes, source code (i.e., .c, .cpp and .h files) is generated. Figure 16 also shows traceability relationships among artifacts produced in the process phases.

XVI.ii. Roles in Our Tailored Harmony Process

The main roles to assign during process execution are at least 7 and they are listed in the following.

1. **Stakeholder**, who represents an interest group whose needs must be satisfied by the project. This role requires specific domain knowledge and may be assigned to one or more staff members, even though such staff also has one of the following roles: Requirements Specifier or Tester.

2. **Project Manager**, who plans, manages and allocates resources to tasks; shapes priorities; coordinates interactions with customers and users; keeps the project team focused; selects tools, documents to use; decides documentation to generate;
deliberates achievement of a goal or accomplishment of a task; schedule meetings, peer reviews and milestone reviews as appropriate. The Project Manager also establishes a set of practices that ensure the integrity and quality of project work products. A Project Manager may also be a Software Architect. Process mastering.

3. **Requirements Specifier**, who specifies and maintains the detailed system requirements, including non-functional requirements and system constraints. S/he specifies and maintains use cases, i.e. Black-Box Use Case Design. S/he understands user needs. The role may be assigned together with the role of Test Designer.

4. **Test Designer**, who identifies, defines and conducts the required tests. This includes identifying and describing the appropriate techniques, tools and guidelines to implement the required tests. The role also represents stakeholders who do not have direct or regular representation on the project. A Test Designer may also be a Software Architect, or a Designer.

5. **Architect**, who has overall responsibility for driving the major technical decisions, expressed as the architecture. This typically includes identifying and documenting the architecturally significant aspects of the system, including requirements, design, implementation, and deployment "views" of the system. The architect is also responsible for providing rationale for these decisions, balancing the concerns of the various stakeholders, driving down technical risks, and ensuring that decisions are effectively communicated, validated, and adhered to. Also, it is responsible for planning the integration. If the project is large enough an architecture team may be created. For smaller projects, a single person may act as both project manager and software architect. In both cases, software architecture is a full-time function, with staff permanently dedicated to it.

6. **Designer**, who leads the design of a part of the system, within the constraints of the requirements, architecture, and development process for the project. S/he identifies and defines the responsibilities, operations, attributes, and relationships of design elements (e.g. Blocks, Activity Diagrams, Sequence Diagrams, Statechart diagrams, Interface and Port definition diagrams). S/he ensures that the design is consistent with the architecture. It is common for a person to act as both implementer and designer, taking on the responsibilities of both roles.

7. **Implementer**, who is responsible for developing and testing components, in accordance with the project's adopted standards, for integration into larger subsystems. When test components, such as drivers or stubs, must be created to support testing, the implementer is also responsible for developing and testing the test components and corresponding subsystems. It is possible for two persons to act as the implementer for a
single part of the system, either by dividing responsibilities between themselves or by performing tasks together, as in a pair-programming approach.

Further roles may be required for specific activities, including:
- Independent quality management.
- Configuration management.
- Change management.
- Process enforcement and monitoring.

Also, specific roles may be required for specialists and experts, e.g. tool specialist, security expert, safety engineer and so on.

However, the selected roles are the trade-off between having many roles and having too many activities assigned to the same role.

**XVI.ii.1. Mapping Between Roles and Activities**

Only a part of existing roles is involved in each activity. When involved in an activity, a role may be either primary (i.e., the role leads the activity and is responsible for decisions) or secondary (i.e., the role is involved in the activity and its presence is mandatory). Mapping between roles and activities is reported in Figure 17, which further details on the specific tasks each role is expected to perform during each activity are reported in Figure 17.

**XVI.iii. Description of Process Activities**

**XVI.iii.1. Analyze Stakeholder Requests**

The objective of the requirements analysis phase is to analyze the process inputs, i.e., stakeholder requests from systems engineers. For the specific context, as stated in “Software Development Process Enhancement: Baseline Document”, input to software engineers is composed of a sort of functional specification in SysML, quite enough incomplete. Non-functional requirements are also provided on the side of functional specification and they are initially incomplete and informal.

Because of this, the work to be performed on the stakeholder requests (i.e. functional and non-functional specification) is considerable. In particular, a step of requirements elicitation is fundamental to come with a set of software requirements which may be used by software engineers. In order to perform requirements elicitation, an overall understanding of the system is mandatory. In the aim of having a shared understanding of the system among software team members, a global analysis of stakeholder requests and needs is performed in the form of a meeting.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Role</th>
<th>Stakeholder specifier</th>
<th>Requirement specifier</th>
<th>Architect</th>
<th>Designer</th>
<th>Implementer</th>
<th>Test Designer</th>
<th>Project Manager</th>
<th>Number of roles involved in the activity</th>
<th>#Primary roles</th>
<th>#Secondary roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze stakeholder requests.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define system validation plan.</td>
<td>P</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify requirements, interfaces, and trace them to stakeholder requests</td>
<td>P</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review requirements, traceability, validation plan, and perform risk analysis.</td>
<td>P</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identify and trace Use Cases to Requirements (Content Diagram and UCD).</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define Black Box Use Case specification (AD+SD+H&amp;D+SD)</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define and trace system test plan.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify black-box UCs through execution and correct problems.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review Black Box Use Cases, system test plan, and reports, and perform risk analysis.</td>
<td>P</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define System Architecture (IBD).</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define White Box Use Case specification (AD+SD+H&amp;D+SD+SDD+H&amp;D)</td>
<td>S</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define system integration test plan.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify white-box use cases through execution and correct problems.</td>
<td>S</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review architecture, White Box Use Cases, system integration test plan, reports, and perform risk analysis.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate or develop tasks and trace to white box use cases.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define and trace unit test plan.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perform unit test, identify bug correction actions.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perform system integration test, identify bug correction actions.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perform system test, identify bug correction actions.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perform system validation.</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
Number of tasks in which the role is involved:

<table>
<thead>
<tr>
<th>Stakeholder specifier</th>
<th>Architect</th>
<th>Designer</th>
<th>Implementer</th>
<th>Test Designer</th>
<th>Project Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

#Primary tasks:

<table>
<thead>
<tr>
<th>Stakeholder specifier</th>
<th>Architect</th>
<th>Designer</th>
<th>Implementer</th>
<th>Test Designer</th>
<th>Project Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

#Secondary tasks:

<table>
<thead>
<tr>
<th>Stakeholder specifier</th>
<th>Architect</th>
<th>Designer</th>
<th>Implementer</th>
<th>Test Designer</th>
<th>Project Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
```
In Harmony, a Stakeholder Requirements Document is generated after analyzing and refining the stakeholder requests. In the context of the Impact-One, it is not necessary to develop an ad hoc document, as stakeholder requests are already formalized in the so called Sub-System Specification (SSS).

The following roles shall be involved in the activity as primary roles:
- Project manager, who plans the meeting, drives the communication and defines the scope of the system
- Stakeholder, who presents its needs and constraints.
- Requirements specifier, who elicits needs.

The following metrics shall be considered:
- Average time spent for analyzing each existing stakeholder requests during the activity (requires the number of stakeholder requests).
- Average time spent for modifying each existing stakeholder requests during the activity (requires the number of modified stakeholder requests).
- Average time spent for adding each stakeholder requests during the activity (requires the number of stakeholder requests before and after the activity).

**XVI.iii.2. Define System Validation Plan**

A plan for validating stakeholder requests should be defined. Agreement upon that plan between systems engineers and project manager is mandatory. Eventually, the validation plan shall include the detailed description of all test scenarios systems engineers are willing to execute and their expected results. The population of the validation plan is iterative, but any modification to that plan shall go through a change management process. In fact, changes to the validation plan may lead to considerable rework on the project, as they may impact requirements and their understanding.

The following shall be involved in the activity as primary roles:
- Stakeholder, who presents its needs for validation.
- Project manager, who accepts and agrees upon the stakeholder validation plan.

The following shall be involved in the activity as secondary roles:
- Requirements Specifier, which participates in the activity for a deeper understanding of stakeholder needs.
- Test Designer, who models test scenarios for validation and create the needed documentation.

The following metrics shall be considered:
• Average time spent for defining each validation test case (requires the number of validation test cases).
• Average time spent for defining all validation test cases relating to a stakeholder request (requires the average number of validation test cases per stakeholder request).

**XVI.iii.3. Specify Requirements and Traceability to Stakeholder Requests**

After analyzing stakeholder requests, the System/Software Requirements document (SRS) shall be generated. Such document formalizes functional and non-functional requirements in the form of “shall statements”. Each statement describes a single functionality of the systems.

Together with SRS, a traceability matrix maps stakeholder requests to system requirements, and a Requirements Diagram models requirements in SysML. Such a diagram helps understanding relationships among requirements (e.g. generalization or derivation), and may support a well-structured traceability policy: for example, whenever a verification tool (e.g. Test Conductor) does not provide functionality for tracing requirements to test cases, SysML requirements (i.e., model elements) may be traced to sequence diagrams modeling test scenarios (still model elements).

Traceability to stakeholder requests, instead, is required to:

1. Check that each stakeholders’ need is addressed in at least one requirement.
2. Each specified requirement is motivated by at least one actual need.
3. Support the change management process.

The following shall be involved in the activity as primary roles:

• Requirements specifier, who formalizes functional and non-functional requirements.

The following metrics shall be considered:

• Average time spent for defining each requirements during the activity (requires the number of requirements).
• Average time spent for tracing each requirement to stakeholder requests (requires the number of traces from requirements to stakeholder requests).
• Average time spent for tracing each stakeholder request to requirements (requires the number of traces from stakeholder requests to requirements).
• Average time spent to define all requirements deriving from a given stakeholder request (requires the number of requirements and the number of stakeholder requests).
• Average time spent on the requirements diagram per added requirement (requirements the number of requirements before and after ending the activity).
XVI.iii.4. Reviews and Risk Analysis

A software review is a “process or meeting during which a software product is examined by a project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval” [14]. Different degrees of formalism for such reviews may be adopted, depending on company process constraints, project constraints and needs.

Along the process, at least the following reviews shall be performed within the project development:

- Review of requirements, traceability, and risk analysis.
- Review Black Box UCs, traceability, system test plan, reports, and risk analysis.
- Review White Box UCs, integration test plan, reports, and risk analysis.

Additional reviews shall be planned whereas the company forces their execution at specific stages of the development. Such additional reviews should not match any of the previously listed three reviews.

Reviews are to be scheduled by the project leader. Details for each review are provided in the following subparagraphs. During each meeting, a role has in charge to lead the review.
In every review, also traceability and risk analysis shall be discussed and checked, and a specific subparagraph describes their review.

**XVI.iii.5. Review of Requirements**

During a requirements review, Requirements Specification Checklist (see later in this Section), Project Constraints and Performance Checklist, and Test Checklist shall be used.

The following shall be involved in the activity as primary roles:

- Project manager, who plans the meeting and deliberates the meeting end when no TODOs or doubts on the documents are left.
- Requirements specifier, who leads the meeting by reading and explaining specified requirements and presenting arisen doubts, issues and risks.
- Stakeholder, who confirms and validates requirements.

The following shall be involved in the activity as secondary roles:

- Architect, who participates in the meeting and understands architecture constraints.
- Designer, who participates in the meeting and understands design constraints.

The following metrics shall be considered:

- Average time spent for reviewing each requirements (requires the number of reviewed requirements).

**XVI.iii.6. Review of Black Box Use Case Design, System Test Plan, and Verification Reports**

To review the system plan means to check whether a sufficient number and types of test cases have been defined and planned for subsequent system test activities (not model execution, but code execution, possibly in the target environment or simulator). “Sufficient” includes at least two test cases for each function, i.e. an expected input and an unexpected input, even though, in general, coverage criteria shall be defined and satisfied by the plan. When such criteria are defined, review the test plan document means to check that coverage criteria are met.

To review the verification report means to check whether: (1) model execution output matches the expected output in the corresponding system test case, and (2) model execution flow matches the expected flow as specified in requirements.

During Black Box Use Case Design review, checklists Use Case Model Checklist, Software Architecture Checklist, and Test Checklist shall be used.

The following shall be involved in the activity as primary roles:

- Project manager, who plans the meeting and deliberates the meeting end when no TODOs or doubts on the documents are left.
• Architect, who leads the meeting by reading the Black Box Use Case Design, and presenting arisen doubts, issues and risks.
• Test Designer, who describes the system test plan, i.e. system test cases associated to Black Box Use Case Design.

The following shall be involved in the activity as secondary roles:
• Requirement specifier, who participates in the review by checking that all requirements are met and constraints satisfied.
• Designer, who participates in the meeting and understands design constraints.

The following metrics shall be considered:
• Average time spent for reviewing each black box use case (requires the number of reviewed black box use case).
• Average time spent for reviewing each diagram of the black box use case design (requires the average number of diagrams in the design).
• Average time spent for reviewing all diagrams of a given black box use case (requires the average number of diagrams per black box use case).
• Average time spent for reviewing each block defined for the black box use case design (requires the number of blocks in the design)
• Average time spent for reviewing all blocks defined for a given black box use case (requires the average number of blocks per black box use case)
• Average time spent for reviewing each system test case (requires the number of test cases).
• Average time spent for reviewing each verification report entry (requires the number verification report entry).

**XVI.iii.7. Review of Architecture, White Box Use Case Design, System Integration Test Plan, and Reports**

To review the system integration plan means to check whether a sufficient number and types of test cases have been defined and planned for subsequent integration test activities (not model execution, but code execution, possibly in the target environment or simulator). “Sufficient” includes at least two test cases for each function, i.e. an expected input and an unexpected input, even though, in general, coverage criteria shall be defined and satisfied by the plan. When such criteria are defined, review the test plan document means to check that coverage criteria are met.
To review the verification report means to check whether: (1) model execution output matches the expected output in the corresponding system integration test case, and (2) model execution flow matches the expected flow as specified in Black Box Use Case Design.

During White Box Use Case Design review, checklists Use Case Model Checklist, Subsystem Design Checklist, and Test Checklist shall be used.

The following shall be involved in the activity as primary roles:
- Project manager, who plans the meeting and deliberates the meeting end when no TODOs or doubts on the documents are left.
- Architect, who check whether provided architecture and constraints are met.
- Designer, who leads the meeting by reading the White Box Use Case Design, and presenting arisen doubts, issues and risks.

The following shall be involved in the activity as secondary roles:
- Test Designer, who describes the integration test plan, i.e. integration test cases associated to White Box Use Case Design.

The following metrics shall be considered:
- Average time spent for reviewing the architecture per architectural block (requires the number of architectural blocks)
- Average time spent for reviewing each white box use case (requires the number of reviewed white box use case).
- Average time spent for reviewing each diagram of a white box use case design (requires the average number of diagrams for each white box use case design).
- Average time spent for reviewing each system integration test case (requires the number of test cases).
- Average time spent for reviewing each verification report entry (requires the number verification report entry).

**XVI.iii.8. Review Traceability Matrices**

Reviewing traceability means to check some properties for all mappings, i.e.:
- Stakeholder requests to requirements.
- Requirements (including non functional) to Use Cases (black box design).
- Use cases (white box) to units and to system integration test cases.
- Units to unit test cases.

Properties to check are:
1. All higher level artifacts have been traced to at least one lower level artifact.
2. All lower level artifacts are traced back to at least one higher level artifact.
3. All trace links are correct, i.e. the traceability makes sense for each trace link.
4. All trace links are complete, i.e. there is no trace link missing from/to an artifact.

The following metrics shall be considered:
- Average time spent for reviewing each trace from the lower level to the higher level (requires the number of traces from the lower level to the higher level).
- Average time spent for reviewing each trace from the higher level to the lower level (requires the number of traces from the higher level to the lower level).

XVI.iii.9. **Analysis of Risks**

A number of aspects and threats should be analyzed during each review. In case something goes wrong, it may turn to have a major impact on project progress, e.g.: much rework to be performed, the project to fail, to overrun budget or schedule, or not to meet stakeholder needs. Aspects and threats to be accounted include at least: achievability and reasonability of performance goals; volatility of requirements; robustness of the architecture with respect to change; mortality of resources; adequate documentation of algorithms and design rationale; process assurance.

In case a structured risk management sub-process is part of the company culture, such plan should be updated and augmented according to review findings.

The following shall be involved in the activity as primary roles:
- Project manager, who drives the analysis and accept mitigation plans and strategies.

Since all other roles, but the stakeholder, are involved in at least one of the reviews, they shall collaborate and contribute, from their specific points of view, to the risk analysis performed during the meetings they participate in.

The following metrics shall be considered:
- Average time spent for analyzing each risk (requires the number of analyzed risks).

XVI.iii.10. **Identify and Trace Use-cases to Requirements**

The first step is to create a use case model which starts with the identification of actors and use cases.

An actor is a model element which represents an entity external to the system under development, i.e. an entity whose development is not part of the current project, but which offers and requires interfaces for interacting with the system under development.

A use case is a model element which captures a required system behavior from the perspective of the end-user. It is not a function of the system, but rather a set of related scenarios. A
scenario is a non-branching sequence of interactions between actors and system under development. Thus, a use case collects all situations a user can find in when trying to achieve a high-level goal the system is required to serve.

Both actors and use cases are part of a system use case diagram, which includes: all use cases, both the one primary actor (i.e. the actor activating the use case) and all secondary actors (i.e. non primary actors involved in the use cases) for each use case, and all relationships among actors (i.e. generalization), among use cases (i.e. extension and inclusion), and between actors and use cases (i.e. association).

![Figure 19 System use case diagram in Rhapsody](image)

Subsequently, a traceability matrix is incrementally built up, which maps requirements to use cases. This traceability flow down supports the change management process, and allows to check whether all requirements have been allocated to use cases.
Also, allocation of requirements can be formalized through the use of the SysML “allocation” relationships. Some development tools, including Rhapsody, automatically create an allocation relationship when the traceability matrix is populated between use cases and requirements.

The following shall be involved in the activity as primary roles:

- Requirements specifier, who performs all tasks.

The following metrics shall be considered:

- Average time spent for identifying each use case (requires the number of use identified cases).
- Average time spent for reviewing each trace from uses cases to requirements (requires the number of traces from uses cases to requirements).
- Average time spent for reviewing each trace from the requirements to uses cases (requires the number of traces from the requirements to use cases).

**XVI.iii.11. Black Box Use Case Design**

The design of Black Box Use Case is the same as in Harmony and may be performed in several ways, i.e. any possible path in the following activity diagram, taken from the Harmony Deskbook v3.1.2.
The first step is the definition of the system context in an *Internal Block Diagram*. The elements of this diagram are instances of SysML blocks, representing each use case and its associated actor(s).
At this stage, blocks are empty and not linked, and their definition is functional to the model-based approach we are willing to apply. Any path in the activity includes the creation of: one activity diagram per use case; a number of sequence diagrams (one per scenario); one statechart diagram. Also, any path includes the definition of ports and interfaces in an internal block diagram associated to the use case. The previous representation of use cases as SysML block allows to automatically and immediately link each use case to its internal structure and behavior.
Figure 24 Sequence Diagram in Rhapsody

Figure 25 Internal Block Diagram in Rhapsody

Figure 26 Statechart diagram in Rhapsody
The output of the Black Box Use Case Design is the construction of an executable model.

The following shall be involved in the activity as primary roles:

- Requirements specifier, who performs all tasks.

The following metrics shall be considered:

- Average time spent for defining each black box use case (requires the number of defined black box use case).
- Average time spent for defining each diagram of the black box use case design (requires the average number of diagrams in the design).
- Average time spent for defining all diagrams of a given black box use case (requires the average number of diagrams per black box use case).
- Average time spent for defining each block defined for the black box use case design (requires the number of blocks in the design).
- Average time spent for defining all blocks defined for a given black box use case (requires the average number of blocks per use case).
- Average time spent for defining all use cases deriving from a given requirement (requires the average number of use cases deriving from each requirement).

XVI.iii.12. Define and Trace System, System Integration, and Unit Test Plans

Based on the gathered and formalized requirements a plan for testing the system shall be defined.

Based on the specified subsystem interfaces, a plan for testing the system integration shall be defined.

Based on the developed units, a plan testing each unit shall be defined.

All plans include all details required to run a test case and check whether the actual output matches the expected output, and whether the actual behavior matches the expected behavior. An appropriate coverage of system cases over system requirements (and of systems integration test over interface requirements, and of unit test over unit requirements) is expected to be achieved through system and system integration test. Criteria for formalizing the term “appropriate coverage” and making it objective are to be defined in advance, also according to system non-functional requirements (e.g. reliability, safety or performance). Also traceability matrices are required: one should trace each requirement to a set of system test cases constituting the system test plan; another one should trace each interface requirements (i.e. each operation) to a set of system integration test cases; one last should trace each unit requirements (i.e. each class method) to a set of unit test cases.

The following shall be involved in the activity as primary roles:
• Test Designer, who defines, details and models system test cases associated to the Black Box Use Case Design and system integration test cases associated to the White Box Use Case Design. He/she is also responsible for generating the required documents.

• Requirements specifier (only for system test plan), who assures that coverage criteria are met by the system test plan for verifying the correct system functionality.

• Architect and Designer (only for system integration test plan), who assures that coverage criteria are met by the system integration test plan for verifying the correct interaction among subsystems. Both Architect and Designer roles are required, because they are both involved in the White Box Use Case Design, even though they operate at different levels of abstraction (higher for the first role, lower for the last role).

• Implementer (only for unit test plan), who assures that coverage criteria are met by the unit test plan for verifying the correct unit functionality.

The following metrics shall be considered:

• Average time spent for defining each system test case (requires the number of system test cases).

• Average time spent for defining all system test cases for a given requirement (requires the average number of system test cases per requirement).

• Average time spent for defining each system integration test case (requires the number of system integration test cases).

• Average time spent for defining all system integration test cases for a given communication link between blocks (requires the average number of system test cases per communication link between blocks).

• Average time spent for defining each system test case (requires the number of unit test cases).

• Average time spent for defining all unit test cases for a given unit (requires the average number of unit test cases per unit).

XVI.iii.13. Verify Black-Box and White Box Use Case Design through Execution

The use case model shall be analyzed through model execution. The primary focus of this form of verification is on checking the generated sequences of messages in response to the given stimuli, rather than on testing the underlying functionality. This last aspect, in fact, will be checked later in the development, as the focus is on verifying the correct interaction between the system under development and the environment.

In order to execute the Use Case Model, execution scenarios shall be defined and run, e.g. through Rhapsody animations. This way, for a given use case and a given scenario, the output
of the execution of the use case model can be matched to the expected black box system behavior definition (i.e. the specified behavior).

Detected problems, e.g. errors, incompleteness, or inconsistencies concerning requirements shall be addressed and fixed.

The following shall be involved in the activity as primary roles:

- Requirements specifier, who executes the model.

The following metrics shall be considered:

- Average time spent for defining each executable models (requires the number of defined executable models)
- Average time spent for defining all executable models for a given use case (requires the average number of executable models per box use).
**XVI.iii.14. Define System Architecture**

Defining the system architecture, in Harmony words, “means to determine the best means of achieving the capability of a particular function in a rational manner”. A way for doing that is applying the weighted objectives method (Cross 1989), which allows to potential solutions to functional problems. Such approach includes a sequence of steps, i.e.:

- Identity key system functions, i.e. sets of related activities defined in the Black Box Use Case model which may be realized by a single architectural component, e.g. “Capture Biometric Data” may group the activities of “scanning biometrics data”, “authenticate biometric data”, and “display authentication status”, which may be a sub-set of the activities of the use case “Control entry to the system”.
- Define a set of candidate solutions for each key system function, e.g. “Fingerprint scanner” and “Optical scanner”.
- Identify assessment (solution) criteria for each key system function.

<table>
<thead>
<tr>
<th>Solution Alternatives</th>
<th>Solution Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td>Fingerprint Scanner</td>
<td></td>
</tr>
<tr>
<td>Optical Scanner</td>
<td></td>
</tr>
</tbody>
</table>

Figure 28 Example of assessment criteria

- Assign weights to assessment criteria, according to their relative importance.

<table>
<thead>
<tr>
<th>Solution Alternatives</th>
<th>Solution Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td>Fingerprint Scanner</td>
<td>Wt = 0.3</td>
</tr>
<tr>
<td>Optical Scanner</td>
<td>Wt = 0.7</td>
</tr>
</tbody>
</table>

Figure 29 Example of weight assignment to assessment criteria
Define utility curves for each assessment criterion, i.e. curves where the x axis is the extent to which an assessment criterion may be met, and the y axis is a measure of effectiveness for that criterion. Some criteria, in fact, have a constantly increasing utility function, i.e. the more, the better (e.g. reliability), while other criteria may have a step function.

![Utility Curve Shapes](image)

Assign measures of effectiveness to candidate solutions for each criterion. The measure of effectiveness is relative value with respect to a nominal value of the corresponding utility curve.

![Assignment of Measures of Effectiveness](image)
• Determine solution by weighted objectives calculation: for each assessment criterion, multiply the measure of effectiveness and its weight, so getting its the weighted value. Add up weighted values for all assessment criteria to produce a score for that candidate solution. The selected solution is the one with the highest score.

<table>
<thead>
<tr>
<th>Solution Alternatives</th>
<th>Accuracy</th>
<th>Purchase Cost</th>
<th>Installation Cost</th>
<th>Maintenance Cost</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MoE</td>
<td>W</td>
<td>MoE</td>
<td>W</td>
<td>MoE</td>
</tr>
<tr>
<td>Fingerprint Scanner</td>
<td>7.6</td>
<td>2.26</td>
<td>7.25</td>
<td>1.45</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optical Scanner</td>
<td>10.0</td>
<td>3.0</td>
<td>3.75</td>
<td>0.75</td>
<td>2.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.316</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7.165</td>
</tr>
</tbody>
</table>

MoE = Measurement of Effectiveness Value  
W = Weighted Value = WT * MoE  
Weighted Total = \( \sum \) W(solution Alternative)

Figure 33 Example of solution determination

• Merge possible solutions to form system architecture. In fact, the previous study on utility curves, assessment criteria, measures of effectiveness and weighted objectives calculation are to be performed for each key system function. Thus, each key system function will generate an architectural solution, which needs to be merged with all architectures generated by analyzing other key system functions. The system architecture is the result of merging such key system function architectures.

The following shall be involved in the activity as primary roles:

• Architect, who makes decisions regarding the architecture, perform architecture analysis (e.g. application of the weighted objectives method), by taking into account both functional and non-functional requirements.

The following shall be involved in the activity as secondary roles:

• Designer, who collaborates to the definition of the architecture by providing feedback on the feasibility of the architecture itself.

The following metrics shall be considered:

• Average time spent for defining each architectural element (requires the number of defined architectural element)

• Average time spent for defining all communication links for a given architectural element (requires the average number of communication links per architectural element).

• Average time to generate all architectural blocks derived from a given use case (requires the number of black box use cases).
XVI.iii.15. White Box Use Case Design

Based on the system architecture concept, the system under development shall be decomposed into parts. Such decomposition shall be documented in Block Definition Diagrams and Internal Block Diagrams and may be performed in a number of subsequent steps, i.e. at gradually lower levels of abstraction. Each black box activity diagram shall be transformed into a white box one by creating swim lanes, each representing a part of the system, and allocating activities to those lanes.

If an activity cannot be allocated to a single block, then it needs to be broken down into at least two activities.

Given white box activity diagrams, white box sequence diagram shall be derived. In such diagrams, interactions among subsystems are modeled. Lifelines in white box sequence diagrams correspond to blocks representing subsystems (or, at a lower level, to parts belonging to subsystem blocks). When decomposing a block into parts, block operations shall be partitioned among parts and possibly decomposed into more operations (in case a single operation cannot be allocated to one part). In this last case, the creation of a dependency among lower level operations and higher level operation is required to keep trace of that decomposition. When an operation does not need to be broken down into more operations, it can be just copied to the part in which it has to be allocated.
Ports and interface shall also be defined, which is made easier when modeling subsystem interaction by means of white box sequence diagrams, where each message or service represents an interface requirement. Once ports and interfaces are defined, the state-base subsystem behaviors are captured into statechart diagrams.

Notice that no explicit traceability is necessary during this activity. In fact, a one-to-one mapping is planned between use cases identified during Black Box Use Case Design and the use cases to be detailed during White Box Use Case Design. However, during review activities, at least completeness shall be verified, i.e. every use case identified during Black Box Use Case Design has been considered during White Box Use Case Design.

During decomposition, architectural and design patterns may be leveraged. For example, Boundary-Control-Entity (Eclipse 2010) pattern may contribute to define flexible, reusable, easy-to-understand and software engineered subsystems.

An entity element is responsible for some meaningful chunk of information. This does not imply that entities are (or store) data. Entities perform meaningful behavior around some cohesive amount of data. If they do not, then they are just “beans”, not “entities”. An entity interacts with other entities or with control elements.

A control element manages the flow of interaction of the scenario. A control element could manage the end-to-end behavior of a scenario or it could manage the interactions between a subset of the elements. A control element interacts with any kind of elements (i.e. boundary, entity or other control). Behavior and business rules relating to the information relevant to the scenario should be assigned to the entities; the control elements are responsible only for the flow of the scenario.

A boundary element lies on the periphery of a system or subsystem, but within it (as opposite to actors, which are outside). For any scenario being considered, either across the whole system or within a subsystem, some boundary elements will be the “border elements” that accept input from outside, and provide output to external elements, according to the interface specification of the system or subsystem they belong to. Sometimes boundary elements match Graphical User Interface elements, but this is not a rule. A boundary element interacts with a control element or those boundary elements strictly correlated to it, i.e. not with those boundary elements managing different scenarios (e.g. interface with other subsystems or a different user interface).

The following shall be involved in the activity as primary roles:

- Designer, who defines subsystems internal structure and behavior, allocate operations, and decompose system blocks.
The following shall be involved in the activity as secondary roles:

- Architect, who ensures that architecture is understood and respected.

The following metrics shall be considered:

- Average time spent for defining each white box use case (requires the number of defined white box use case).
- Average time spent for defining each diagram of the white box use case design (requires the average number of diagrams in the design).
- Average time spent for defining all diagrams of a given white box use case (requires the average number of diagrams per black box use case).
- Average time spent for defining each block defined for the white box use case design (requires the number of blocks in the design)
- Average time spent for defining all blocks defined for a given white box use case (requires the average number of blocks per use case)

**XVI.i.16. Develop or Generate Units and Trace to White Box Use Case Design**

In a model-based approach, it is possible to generate code from design models. Specifically, it should be possible to generate source code from UML classes. When source code cannot be completely generated, a hybrid approach is also allowed, where code is partially generated and then manual coding is required to completely implement class behaviors. In general, to develop a unit may also be seen as a task of high level. For example a unit may be a system in a system-of-systems. In this case, after applying all process activities at system-of-systems-level, development of a unit may be demanded to a system development team. White Box Use Case Design, in this case, represents the set of stakeholder requests to provide to the system development team. Thus, that team may start its development process, possibly applying the same process, but a lower level of abstraction and complexity.

However, in both situations, i.e. code to be written or generated, and a system to be developed, each unit will need to trace some design elements. Specifically, each unit shall have at least one use case to refer to. Conversely, each block or part of the White Box Use Case Design shall be implemented by at least one unit.

The following shall be involved in the activity as primary roles:

- Implementer, who writes or generates unit code and trace units to model elements of the White Box Use Case Design.

The following metrics shall be considered:

- Average time spent for generating each leaf unit (i.e. code) (requires the number of generated units).
• Average time required to generate all units deriving from a given use case (requires the average number of units per use case).

**XVI.iii.17. Perform Unit Test, System Integration Test, and System Test**

Plans for unit test, system integration test, and system test already exist at this stage of the process, as they were previously defined. The test case execution can be automatic, if a test environment (partially) support that, or manual. In the last case, drivers and stub may be required.

When a bug is detected, a correction action shall be identified and their impact on existing system estimated. If required, a problem report may be generated and provided as input to the change management process. If the bug correction does not impact any other part of the system, it can be turned into practice and the test is run one more time, possibly with leveraging regression testing.

The following shall be involved in the activity as primary roles:

• Test Designer, who executes test cases, matches actual output against expected output and documents results.

The following shall be involved in the activity as secondary roles:

• Implementer (only for unit test), who provides support to unit test execution and analysis.
• Architect and Designer (only for system integration test), who provide support to system integration test execution and analysis.
• Requirements specifier (only for system test), who provides support to system test execution and analysis.
• Project Manager (only for system test), who deliberates when the system meets all requirements and is ready to go to validation.

The following metrics shall be considered:

• Average time spent for running all test cases for a given unit/communication link/requirements (requires the number of test cases).
• Average time spent for fixing a bug detected during unit/system integration/system test (requires the number of detected bugs).

**XVI.iii.18. Perform System Validation**

System validation is a formal meeting in which the stakeholder receives a work product that is expected to satisfy its needs. During validation, stakeholders base on the system validation plan
to run scenarios. The correct execution of each of those scenarios is mandatory to finalize project development.

The following shall be involved in the activity as primary roles:

- Project Manager, who leads the activity and shows how the product fulfills stakeholder requirements.
- Stakeholder, who checks whether the product meets its needs and what planned in the system validation plan.

The following metrics shall be considered:

- Average time spent for running all validation test cases for a given stakeholder request (requires the number of validation test cases).
- Average time spent for fixing a bug detected during validation (requires the number of detected bugs).

**XVI.iii.19. Further Activities**

Most project management activities are hard to fit into specific steps of the proposed development process, as most of them are asynchronous or constantly ongoing during project development. Such activities include at least:

1. assign roles to resources;
2. plan, manage and allocate roles/resources to tasks;
3. shape priorities;
4. coordinate interactions with customers and users;
5. select tools to use;
6. establish formal documents to generate;
7. deliberate achievement of a goal
8. deliberate accomplishment of a task;
9. schedule meetings, peer reviews and milestone reviews as appropriate.

**XVI.iv. Artifacts and Tools**

During process execution, a number of artifacts shall be generated, including text documents, SysML and UML documents, and test cases. Furthermore, different development environments may be used to support the creation of such artifacts. Specifically, 3 kinds of tools were identified: Requirements Management Tool, Model-Based Development Tool, and Verification Environment.

Figure 35 traces: (a) artifacts to process phase in which it is generated; (b) generated artifacts to sub-folders (or equivalent structures) in which they should be stored within the development environments; (c) each artifact to the kind of tool expected to manage it.
<table>
<thead>
<tr>
<th>Process Phase</th>
<th>Generated Artifacts</th>
<th>Tool Sub-folders</th>
<th>Stakeholder Requests</th>
<th>Requirements Specification</th>
<th>System Validation</th>
<th>System Use Cases</th>
<th>Black-Box: UCDesign (1 folder/UC)</th>
<th>System Verification</th>
<th>Architecture</th>
<th>White-Box UC Design (1 folder/UC)</th>
<th>System Integration Verification</th>
<th>Implementation</th>
<th>Unit Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Requirements Analysis</td>
<td>Stakeholder Requests</td>
<td>RMT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Req. spec Document</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Traceability Matrix Struct-Req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System Validation Plan &amp; Test Cases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Functional Analysis</td>
<td>Test Diagram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System Use Case Diagram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Traceability Matrix Req-Black Box UC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use case model context</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Black Box AD, SD, IBP, SC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Black-Box Execution Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System Test Plan &amp; Cases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Design Synthesis</td>
<td>Architecture BDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>White Box AC, SD, IBP, SC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Traceability Matrix Black Box UC-White Box WC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>White Box Execution Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System Integration Plan &amp; Test Cases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit Implementation and Test</td>
<td>Unit Development Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Traceability Matrix Black Box Unit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unit Test Plan &amp; Cases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unit test report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration Verification</td>
<td>System Integration test report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System verification</td>
<td>System test report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System validation</td>
<td>System Validation Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* RMT = Requirements Management Tool (e.g. DCORS)  
* MBDT = Model-Based Development Tool (e.g. Rhapsody)  
* VE = Verification Environment (e.g. Test Conductor)
XVI.iv.1. **Checklists: a Support Tool for Reviews**

During reviews (e.g. Requirements, Test Plan or Design reviews) the use of checklists may support the verification task addressing those system aspects which frequently represent sources of problems or malfunctioning. In the next paragraphs some checklist mainly coming from RUP are presented and suggested for use during process reviews.

**Requirements Specification Checklist**

Check whether the SRS exhibit fundamental characteristics, including:

1. **Correct:** Is every requirement stated in the SRS one that the software should meet?
2. **Unambiguous:** Does each requirement have one, and only one, interpretation? Has the customer's language been used? Have diagrams been used to augment the natural language descriptions?
3. **Complete:** Does the SRS include all significant requirements, whether related to functionality, performance design constraints, attributes, or external interfaces? Have the expected ranges of input values in all possible scenarios been identified and addressed? Have responses been included to both valid and invalid input values? Do all figures, tables and diagrams include full labels and references and definitions of all terms and units of measure? Have all TBDs been resolved or addressed? For each error or exception, a policy defines how the system is restored to a "normal" state. For each possible type of input error from the user or wrong data from external systems, a policy defines how the system is restored to a "normal" state.
4. **Consistent:** Does this SRS agree with the Vision document, the use-case model and the Supplementary Specifications? Does it agree with any other higher level specifications? Is it internally consistent, with no subset of individual requirements described in it in conflict?
5. **Ability to Rank Requirements:** Has each requirement been tagged with an identifier to indicate either the importance or stability of that particular requirement? Have other significant attributes for properly determining priority been identified?
6. **Verifiable:** Is every requirement stated in the SRS verifiable? Does there exist some finite cost-effective process with which a person or machine can check that the software product meets the requirement?
7. **Modifiable:** Are the structure and style of the SRS such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style? Has redundancy been identified, minimized and cross-referenced?
8. **Traceable:** Does each requirement have a clear identifier? Is the origin of each requirement clear? Is backward traceability maintained by explicitly referencing earlier
artifacts? Is a reasonable amount of forward traceability maintained to artifacts spawned by the SRS?

**Project Constraints and Performance Checklist**

1. Have all requirements derived from existing standard and regulations been specified? How will this be traced?
2. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s), etc.?
3. What is the software supposed to do? This should include the following:
   a. Validity checks on the inputs
   b. General responses to abnormal situations, including: overflow, communication facilities, error handling and recovery
   c. Effects of parameters
   d. Relationship of outputs to inputs, including input/output sequences and formulas for input to output conversion
4. What is the speed, availability, response time, recovery time of various software functions, etc.? Are both static and dynamic requirements included?
5. What are the reliability, availability, portability, correctness, maintainability, security, etc. considerations?
6. All nominal and maximal performance requirements are specified.
7. Performance requirements are reasonable and reflect real constraints in the problem domain.
8. The workload analysis model provides estimates of system performance that indicate which performance requirements, if any, are risks.
9. Bottleneck objects have been identified and strategies defined to avoid performance bottlenecks.
10. Expand Collaboration message counts are appropriate given the problem domain.
11. Executable start-up (initialization) is within acceptable limits as defined by the requirements.

**Use Case Model Checklist**

1. Have you found all the actors?
2. Is each actor involved with at least one use case?
3. All functional requirements are mapped to at least one use case.
4. All non-functional requirements that must be satisfied by specific use cases have been mapped to those use cases.

**Software Architecture Checklist**

1. The system has a single consistent, coherent architecture.
2. The number and types of component is reasonable.
3. The system will meet its availability targets.
4. The architecture will permit the system to be recovered in the event of a failure within the required amount of time.
5. The architecture provides defines clear interfaces to enable partitioning for parallel team development.
6. System performance estimates have been validated using architectural prototypes, especially for performance-critical requirements.
7. The architecture provides for recovery in the event of disaster or system failure.

**Subsystem Design Checklist**

1. The name of each subsystem is unique and descriptive of the collective responsibilities of the subsystem.
2. The subsystem description accurately reflects the collective responsibilities of the subsystem.
3. The subsystem, through its interfaces, presents a single, logically consistent set of services.
4. The subsystem is the responsibility of a single individual or team.
5. The subsystem realizes at least one interface.
6. The interfaces realized by the subsystem are clearly identified and the dependencies are correctly documented.
7. The subsystem's dependencies on other model elements are restricted to interfaces and packages to which the subsystem has a compilation dependency.
8. The information needed to effectively use the subsystem is documented in the subsystem facade.
9. Other than the interfaces realized by the subsystem, the subsystem's contents are completely encapsulated.
10. Each operation on an interface realized by the subsystem is utilized in some collaboration.
11. Each operation on an interface realized by the subsystem is realized by a model element (or a collaboration of model elements) within the subsystem.
12. Subsystem partitioning done in a logically consistent way across the entire model.
13. The contents of the subsystem are fully encapsulated behind its interfaces.

**Test Checklist**

1. Each test case states the expected result and method of evaluating the result.
2. Test cases have been identified to execute all product requirement behaviors in the target-of-test.
3. For each requirement for test, at least two test cases have been identified.
4. Each project requirement (as stated in use cases or the supplemental specifications) has at least one associated requirement for test or a statement justifying why it is not a requirement for test.
5. A clear and concise test strategy is documented for each type of test to be implemented and executed. For each test strategy, the following information has been clearly stated: (1) The name of the test and its objective; (2) A description of how the test will be implemented and executed; (3) A description of the metrics, measurement methods, and criteria to be used evaluate the quality of the target-of-test and completion of the test.
6. Have sufficient tests been implemented to achieve acceptable test coverage?

**XVI.iv.2. Contribution of Agile Models to the Proposed Development Process**

An aspect to consider when developing systems is that, rather than depending on a contract or a statement of work, the customer could work closely with the development team, providing frequent feedback. In our specific context, customers are represented by systems engineers, who ask for a subsystem to be developed. Interaction between systems and software engineers may represent the key to the project success. Moreover, direct communication among team members should be frequent enough to share knowledge on the full software system and report encountered issues.

One more remarkable point is that effectively responding to changes requires to make sure that planning is flexible and ready to adapt to those changes. Furthermore, when a change arrives, the development team should come up with a reliable impact analysis, which may be provided only in presence of a clear traceability form stakeholder requirements to code artifacts, passing by all design and intermediate artifacts. Not only change management, but, more in general, a well-structured configuration management leads to a traceable, manageable and predictable evolution of the project. The definition of a well structure configuration management process is scheduled in next iterations of the enhancement process.

**XVI.v. Requirements Engineering Discipline Proposal**

So far we have described the process activities for our tailored Harmony Process and provided a detailed description concerning the metrics to target our goal: project time control. However, that goal is not the only aspect to account for; as an example, in this paragraph, we provide additional metrics for requirements-related activities (i.e. Requirements Engineering Discipline).
<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Definition</th>
<th>Metric</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unambiguous</td>
<td>An SRS is U iff every r in R has one interpretation.</td>
<td>( U = \frac{n_{UI}}{n_r} ) = percentage of requirements with unique interpretation given by reviewers during a review.</td>
<td>( n_{UI} ) = number of requirements with unique interpretation given by reviewers during a review. Ambiguity is function of each reader’s background.</td>
</tr>
<tr>
<td>Complete</td>
<td>An SRS is C iff every known requirement is documented.</td>
<td>( C = \frac{n_r}{n_a+n_b+n_c+n_d} ) = percentage of requirements well documented in the SRS.</td>
<td>( n_a ) = number of known and captured requirements. ( n_b ) = number of requirements poorly specified, abstractly stated, or not yet validated. ( n_c ) = number of known and not yet specified requirements. ( n_d ) = number of requirements we do not understand enough to specify them.</td>
</tr>
<tr>
<td>Correct</td>
<td>An SRS is Ct iff every requirement represents something which fulfills stakeholder needs.</td>
<td>( Ct = \frac{n_C}{n_r} = \frac{n_C}{n_C+n_I} ) = percentage of correct requirements. ( Ct = \frac{n_C}{n_r} = \frac{n_C}{n_C+n_{NV}} ) = percentage of validated requirements.</td>
<td>( n_C ) = number of correct requirements. ( n_I ) = number of incorrect requirements. ( n_I ) needs to be estimated. ( n_{NV} ) = number of not validated requirements. A requirement is validated by the customer.</td>
</tr>
<tr>
<td>Understandable</td>
<td>An SRS is Un iff all classes of SRS readers can easily comprehend the meaning of all requirements.</td>
<td>( Un = \frac{n_{UR}}{n_r} ) = percentage of understood requirements.</td>
<td>( n_{UR} ) = number of understood requirements. An requirement is understood if a reader reads and correctly comments on it to the SRS.</td>
</tr>
<tr>
<td>Requirements with a minimum of explanation.</td>
<td>Consistent</td>
<td>An SRS is Co iff and only if no requirement stated therein conflicts with any project documentation.</td>
<td>[ C_0 = \frac{n_C}{n_r} = \frac{n_C}{n_C + n_I} = \text{percentage of requirements that are consistent with all other documents.} ]</td>
</tr>
<tr>
<td>Executable / Interpretible / Prototypable</td>
<td>An SRS is EIP iff and only if there exists a software tool capable of inputting the SRS and providing a dynamic behavioral model.</td>
<td>[ EIP = \frac{n_{EIP}}{n_{r}} = \text{percentage of executable, interpretible or prototypable requirements.} ]</td>
<td>[ n_{EIP} = \text{number of executable, interpretible or prototypable requirements.} ]</td>
</tr>
<tr>
<td>Precise</td>
<td>An SRS is P iff (a) numeric quantities are used whenever possible, and (b) the appropriate levels of precision are used for all numeric quantities.</td>
<td>[ R = \frac{n_p}{n_r} = \text{percentage of precise requirements.} ]</td>
<td>[ n_p = \text{number of precise requirements.} ]</td>
</tr>
</tbody>
</table>

Additionally, the following metrics shall be collected for project progress monitoring:

- Missing boundaries: based on the BCE architectural pattern, every actor communicates with the system through boundary class. The number of specified missing defined
boundaries provides a rough estimate if the user interface requirements that has not yet been defined in the model.

- Concrete use case not defined: number of concrete use cases with no diagrams as description.
- Hidden model elements: number of model elements not appearing in any diagrams.
- Number of requirements not traced to use cases.

XVI.vi. Analysis and Packaging of Results

The methodology is made available in the form of text descriptions and pictures, while Requirements metrics are in the form of a table. The remaining work products are made available in the form of both text description and IBM Rational Team Concert files, produced by using IBM Rational Method Composer, i.e. multimedia browsable HTML files. Specifically, such work products are:

   a. Flow diagram representing the sequence of activities.
   b. Description of activities.
   c. Description of roles.
   d. Mapping between roles and activities.
   e. Identification of artifacts and support tools.
   f. Mapping between activities, artifacts and tools.

2. Requirements Engineering Discipline.

3. Traceability Discipline.

4. Verification and Validation Milestones.

Snapshots from the HTML documentation are provided in the following.
**Figure 36 Work Breakdown Structure and additional information about the process**
Iteration: Design Synthesis Iteration [n]

Workflow of activities of a given phase

- Define system architecture
- Define and trace system integration test plan
- White box use case design
- Verify white box use case design through execution
- Review architecture, white box use case design, system integration test plan, reports, and analyze risks

Reviewed and Approved Architecture, White Box Use Case Description, Traceability, System Integration Test Plan, Reports and Risks

Figure 37 Workflow of activities of a given phase
Activity: Define system architecture

Figure 38 Activity details

Role Set: Basic roles for Model-Based process

Figure 39 Roles belonging to the defined role set
Role Architect

Role Details: Basic roles for Model-based process

**Relationships**

**Additionally Performs**
- Key Task
- Review block use case design, architecture, traceability, system test plan, reports, and analyze risks
- Verify, white box use case design through execution
- White box use case design

**Modifies**
- Architecture Diagram
- System Integration Test Cases
- System Integration Test Plan
- Traceability matrix White Box Use Case - System Integration Test Case
- White Box Activity Diagram
- White Box Execution Overview
- White Box Interfaces and Ports Definition Diagram
- White Box Sequence Diagram
- White Box Statechart Diagram

**Process Usage**
- Model Based Projects -> System Functional Analysis -> System Functional Requirements Iteration (n) -> Review block use case design, architecture, traceability, system test plan, reports
- Model Based Projects -> Design Synthesis -> Design Synthesis Iteration (n) -> Define system architecture
- Model Based Projects -> Design Synthesis -> Design Synthesis Iteration (n) -> White box use case design
- Model Based Projects -> Design Synthesis -> Design Synthesis Iteration (n) -> Verify, white box use case design through execution
- Model Based Projects -> Design Synthesis -> Design Synthesis Iteration (n) -> Review architecture, white box use case design, system integration test plan, reports, and analyze risks
- White Box Use Case Diagram

**Main Description**

It has overall responsibility for driving the major technical decisions, represented the architecture. This typically includes identifying and documenting the architecturally significant aspects of the system, including requirements, design, implementation, and deployment 'views' of the system. The architect is also responsible for providing rationale for these decisions, following the outcomes of the various stakeholders, driving down technical risks, and ensuring that decisions are effectively communicated, rationalized, and adhered to. Also, it's responsible for planning the integration. If the project is large enough, an architecture team may be created. For smaller projects, a single person may act as both project manager and software architect. In both cases, software architecture is a full-time function, with staff permanently dedicated to it.

**Artifact: Use Case Model Context**

- Work Product/Kind: SysML/VA, wordproduct for Model-Based process

**Relationships**

**Tools**
- Black box use case design
- Define and trace system test plan
- Define system architecture
- Review black box use case design, architecture, traceability, system test plan, reports, and analyze risks

**Process Usage**
- Model Based Projects -> System Functional Analysis -> System Functional Requirements Iteration (n) -> Identify use cases and trace them to requirements
- Use Case Model
- Model Based Projects -> Design Synthesis -> Design Synthesis Iteration (n) -> Define system architecture

Figure 40 Details and description of a role

Figure 41 Details of an artifact

Chapter XVII. Testing Activities: A Complete Example of Activity Characterization

So, we now have an overview of the process and some documentation. However, there is something beyond that description, which cannot actually be modeled easily and completely in the current version of the IBM Rational product we mentioned: the attributes of the impact vectors, which represent the performance of the development activities. The definition of such attributes is not immediate and effortless, but it requires some work we want to make explicit at least for some of those activities. In the rest of this chapter, we report on the careful literature review we performed for testing, including how we carried on the survey and some of its findings. Afterwards, we will end up having a remarkable set of testing attributes and empirical data that allow making decisions on testing.

Over the past decades, hundreds of empirical studies on testing have been performed. Most studies compare a new proposed technique against some others (e.g. against random testing or against similar techniques), while some investigate and assess existing techniques. The way experimental comparisons are carried on, not only for testing, has already been object of study recently and recommendations were provided [15, 16, 17, 18, 19, 20]. Therefore, we decided to carry on an extensive survey of the literature over the last decade in order to identify factors of testing that have a detectable impact on testing performance (in the most general interpretation of this term). The goal of our survey is to define impact vectors, i.e. a homogeneous representation of testing techniques that can enable their comparison and, most importantly, their selection and combination in a practical context.

During our survey, we analyzed more than 2000 papers published between 2004 and 2013 and identified over 200 studies reporting results from the empirical investigation of dozens of testing techniques. In order to produce knowledge useful for the Impact Vector Space construction, we created a tree composed of 192 nodes that characterize a number of points of interest. Examples of nodes high in the tree hierarchy include: resources (e.g. cost and people), performance (e.g. effectiveness and machine time), technology (e.g. test case generation approach and level of automation of test case generation), and domain (e.g. real time systems and database). The leaves of this tree represent the dimensions of the impact vectors, while the non-leaf nodes support organizing dimensions in orthogonal categories (see Figure 42 and Figure 43); each leaf has its own scale (ratio, ordinal or nominal, depending on the nature of the attribute). Some of those techniques can be assigned a priori, as they serve as description of the technique (e.g. is the technique random? Is it classifiable as model-based? Is it coverage based?), but some other characteristics must be derived from existing empirical studies.
Each testing technique is represented as a single impact vector, whose dimension values are assigned based on the data coming from the empirical studies performed on those techniques. Obviously, no study explores all the dozens of factors we identified (nor it reasonably could). However, most studies block a remarkable number of those factors; e.g. the domain is pretty much fixed in every study and can be easily inferred, once defined an adequate scale (nominal) for that attribute. Some other factors will remain unfilled for a technique, because nobody has studied them yet; this will be a gap in the repository of impact vectors concerning that test technique, which future studies will be able to fill.

Figure 42 Excerpt of tree: "Technology" sub-tree

Figure 43 Excerpt of tree: "Performance" sub-tree
One of our findings, however, is that it is not useful to compare absolute values among different studies; e.g. if a technique T1 detects 50% of defects, while another technique T2, in a different study, performs 90%, it does not necessarily mean that T2 is better than T1, e.g. because they were experimented on artifacts of remarkably different quality. Consequently, our approach was to consider absolute numbers only in within the same study; to clarify, if T1 and T2 are techniques investigated within the same study and the percentages of removed defects are 50% and 90%, then T2 results better than T1 and we store this information into a dimension of the impact vector (the scale for such dimension is ordinal and composed of three values: T1 better than T2, T2 better than T1, and T1 equals to T2). The relative goodness of a technique with respect to another is the information stored in the impact vector. As a consequence, the number of dimensions is very high: one dimension exists for each possible non ordered pair of testing techniques that were experimentally investigated.

Obviously, more than one study can compare T1 and T2, but we want to record a single score for each. In order to do that, we identified a number of characteristics [17, 18], an analytical formula to come up with a single-study score, and one more analytical formula that aggregates single-study scores to obtain a global score representing the value of the dimension corresponding to the pair of testing techniques under investigation.

In the following tables, we report the list of characteristics of the empirical study that we account for the definition of the single-study score, along with characteristic definitions and additional information. Notice that such characteristics do not match the attributes of the impact vectors (i.e., the dimension of the impact vector space), but they are only the means through which we valorize the impact vector dimensions (i.e. one per pair of testing techniques).

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference ID</td>
<td>Identification of the experimental study material and results</td>
</tr>
<tr>
<td>Technique T1</td>
<td>A technique value from the mapping table below</td>
</tr>
<tr>
<td>Technique T2</td>
<td>A technique value from the mapping table below</td>
</tr>
<tr>
<td>Slight variation of T1</td>
<td>If (T1=T2) then (slight variation 1 or slight variation 2) = (yes or tool)</td>
</tr>
<tr>
<td>Slight variation of T2</td>
<td>If (T1=T2) then (slight variation 1 or slight variation 2) = (yes or tool)</td>
</tr>
<tr>
<td>Family of T1</td>
<td>A family value from the mapping table below</td>
</tr>
<tr>
<td>Family of T2</td>
<td>A family value from the mapping table below</td>
</tr>
<tr>
<td>Effort/Cost of resources</td>
<td>Result of the comparison (i.e. T1&gt;T2, T1&lt;T2 or T1=T2)</td>
</tr>
<tr>
<td>Machine Execution Time</td>
<td>Result of the comparison (i.e. T1&gt;T2, T1&lt;T2 or T1=T2)</td>
</tr>
</tbody>
</table>

Table 2 Characteristics of the empirical studies to assign a single-study score
<table>
<thead>
<tr>
<th>Defect Detection Capability</th>
<th>Result of the comparison (i.e. T1&gt;T2, T1&lt;T2 or T1=T2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complementary defect types</td>
<td>If &gt;50% defects found are different then high. If 20-50% then partial. Else no/low. In doubt between 2 ranges, be conservative (lower range). If no characterization of defects, then unknown</td>
</tr>
<tr>
<td>Study maturity</td>
<td>Laboratory study, Formal analysis, Laboratory replication or Field study (for further details, see Juristo&amp;al, &quot;Reviewing 25 years of testing experiments&quot;)</td>
</tr>
<tr>
<td>Representativity of objects</td>
<td>Low if toy, high if industrial. If mix of objects, then if &gt;60% industrial then high, else low. If no characterization of objects, then unknown. If objects under study belongs to a unique category, then the field notes has to be filled</td>
</tr>
<tr>
<td>Representativity of subjects</td>
<td>Low if students are not characterized as high skilled, high if professionals or selected skilled students. If mix of subjects, then if &gt;60% professionals then high, else low. If no characterization of subjects, then unknown.</td>
</tr>
<tr>
<td>Representativity of faults</td>
<td>Low if faults are seeded, high if real faults. If mix of defects, then if &gt;60% professionals then high, else low. If no characterization of faults, then unknown. If faults addressed are of a particular type, then the field notes has to be filled</td>
</tr>
<tr>
<td>Type of metric for effort/cost</td>
<td>Direct if: person-hour of similar, $ or similar. Else indirect (e.g. calendar time)</td>
</tr>
<tr>
<td>Type of metric for machine execution time</td>
<td>Direct if: seconds on a given machine, number of instructions, number of operations. Else indirect (e.g. number of test cases)</td>
</tr>
<tr>
<td>Type of metric for Defect Detection Capability</td>
<td>Direct if: found/missed defects. Else indirect (e.g. coverage, killed mutants)</td>
</tr>
<tr>
<td>Notes</td>
<td>Optional (except when representativity of objects or faults is different from high, low or unknown)</td>
</tr>
<tr>
<td>Family</td>
<td>Technique</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>randomly generated test cases</td>
<td>pure random</td>
</tr>
<tr>
<td></td>
<td>manual random</td>
</tr>
<tr>
<td></td>
<td>feedback directed</td>
</tr>
<tr>
<td>functional (black-box) testing</td>
<td>boundary value analysis</td>
</tr>
<tr>
<td>control-flow (white-box) testing</td>
<td>code (statement) coverage</td>
</tr>
<tr>
<td></td>
<td>state chart coverage</td>
</tr>
<tr>
<td></td>
<td>decision (branch) coverage</td>
</tr>
<tr>
<td></td>
<td>condition coverage</td>
</tr>
<tr>
<td>data-flow testing</td>
<td>all-c-uses</td>
</tr>
<tr>
<td></td>
<td>all-p-uses</td>
</tr>
<tr>
<td></td>
<td>all-uses</td>
</tr>
<tr>
<td></td>
<td>all-edges</td>
</tr>
<tr>
<td></td>
<td>all-dus</td>
</tr>
<tr>
<td>mutation testing</td>
<td>strong (standard) mutation</td>
</tr>
<tr>
<td></td>
<td>10% mutation (constrained)</td>
</tr>
<tr>
<td></td>
<td>abs/ror mutation (constrained)</td>
</tr>
<tr>
<td></td>
<td>st-weak/1</td>
</tr>
<tr>
<td></td>
<td>exweak/1</td>
</tr>
<tr>
<td></td>
<td>bb-weak/1</td>
</tr>
<tr>
<td></td>
<td>bb-weak/n</td>
</tr>
<tr>
<td>regression testing</td>
<td>random</td>
</tr>
<tr>
<td></td>
<td>retest-all (traditional)</td>
</tr>
<tr>
<td></td>
<td>Deja-Vu (safe)</td>
</tr>
<tr>
<td></td>
<td>Test tube (safe)</td>
</tr>
<tr>
<td></td>
<td>modification-based / textual differencing</td>
</tr>
<tr>
<td></td>
<td>dataflow/coverage-based</td>
</tr>
<tr>
<td>improvement testing</td>
<td>minimization</td>
</tr>
<tr>
<td></td>
<td>selection</td>
</tr>
<tr>
<td></td>
<td>augmentation</td>
</tr>
<tr>
<td></td>
<td>prioritization</td>
</tr>
<tr>
<td>model-based testing</td>
<td>model-based testing</td>
</tr>
<tr>
<td>concolic testing</td>
<td>concolic testing</td>
</tr>
<tr>
<td>exploratory testing</td>
<td>exploratory testing</td>
</tr>
<tr>
<td>search-based/genetic algorithm testing</td>
<td>search-based/genetic algorithm testing</td>
</tr>
</tbody>
</table>
Given two techniques T1 and T2 investigated in a study, Single Study Score (SSS) is computed with the following formula of our definition: $SSS(T1,T2)=M*R*SO$, where M is the maturity, R is the study relevance, and SO is the study output.

M is 0.5 for laboratory studies, 0.6 when the laboratory study includes a formal (statistical) analysis, 0.75 for laboratory replication, and 1 for field studies. See the mentioned “Reviewing 25 years of testing experiments” by Juristo & al. for further details.

The value of R is 1.5 when the study is of high relevance, 1 for medium relevance, and 0.5 for low relevance. The classification between low, medium and high relevance is made according to the following rules, based on the experiment characteristics reported in Table 4: relevance is high if all applicable characteristics are high/real/direct; relevance is medium if the performance measures are direct and at least 60% of applicable characteristics is high/real; relevance is low in all other cases.

Table 4 Experiment characteristics

<table>
<thead>
<tr>
<th>Accounted characteristics</th>
<th>Sample size</th>
<th>Subjects’ average experience</th>
<th>Artifact type</th>
<th>Faults</th>
<th>Performance measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>Low (&lt;15 subjects) or high (&gt;=15 subjects)</td>
<td>low(&lt;3 years) or high (&gt;=3 years)</td>
<td>Artificial or real</td>
<td>Artificial or real</td>
<td>Proxy or direct</td>
</tr>
</tbody>
</table>

Finally, SO is 1 when T1 performs better than T2, -1 when T2 performs better than T1 and 0 when they perform the same. On this value, we need to add some thoughts: what if T1 performs better, e.g. in terms of effectiveness, but T2 performs better in terms of efficiency? The answer is immediate: for each performance attribute, we produce a different score. Therefore, the score refers only to efficiency or only to effectiveness. Consequently, we have as many dimensions in the impact vector space as the number of performance characteristics we want to consider multiplied by the number of possible pairs of testing techniques.

Based on this formulation, for each study we can obtain a SSS per testing technique performance characteristic. However, as previously stated, for each pair of testing techniques and for each performance characteristics we want to have a single global score. We obtain such Global Score (GS) through the following formula: $GS(T1,T2) = \frac{\sum SSS(T1,T2)}{#studies}$, i.e. the average value of SSSs.

As an example, we report the impact vector generated by accounting only the results of the study in [21] on state chart coverage vs code coverage. In that study, the output is that the two techniques perform the same in terms of percentage of removed defects, so the value for that dimension (relative goodness in terms of effectiveness) is “T1 equals to T2”). Machine time is accounted for, so the value of the corresponding dimension is “Unknown”. However, a lower
level of detail is considered concerning the effectiveness: the authors of [21] investigate the
type of defects detected by each technique. As reported in Table 2, we have a specific set of
dimensions taking that into account. Also for this dimension, there is an issue: different studies
may use different defect taxonomies. However, something more objective is the
complementarity of techniques: if a study detects that two techniques find different defects,
then this is the information we want to store in our impact vector. In particular, in [21], authors
find that combining state chart coverage and code coverage improves the effectiveness of the
verification process, as they detect different defects; consequently, combining them leads to a
higher percentage of detected defects. This information is stored in the “complementary”
dimension of the impact vector (its scale is “high”, “low/no” and “unknown”; objective criteria
to select the value are also defined).

One can easily visualize the results of this review in the form of a square matrix, where rows
and columns are the techniques, while the cells contain the global scores for each pair (rows vs
columns). The matrix is symmetric and the cells on the diagonal do not make sense. An
additional visualization help is the color: since complementarity of defects is very relevant, we
color the cells in yellow when detected defects are of different nature, blue when they are of
the same nature, and white when no information on the defect nature is available. An excerpt
of the matrix we generated is reported in Figure 44.

One more aspect to consider is that if technique T1 performance better (for some performance
attribute) than T2 and T2 performs better than T3, then we can also say that T1 performs better
than T3. In this case, we do not have any direct empirical studies for the pair <T1, T3>, but we
can still insert something in the matrix, even though no scores can be computed; therefore, we
insert a plus or minus sign when we can say, by transitivity, that the technique on the row is
better or worse, respectively, than the one on the column.

So far, we identified around 40 impact vectors, with a total of around 400 pair-wise
comparisons across all testing techniques we analyzed and listed in Table 3. For each impact
vector (technique), we gave value to each dimension which was accounted for in the study or
that could be objectively inferred. Slight variations of a technique are still accounted as a single
technique and such singularity is also stored in an ad hoc dimension of the impact vector.

Summarizing, in order to build an empirically-founded baseline of performance of testing
techniques, we had to go through a long examination of the literature. Without the use of the
Impact Vector Model, the knowledge acquired during such examination would have been
written in a report or, worst case, kept implicit by the people performing the review. Via Impact
Vectors, not only the knowledge is made explicit, but it is also well-structured, machine-
executable, maintainable, and reproducible.
One could argue that this procedure is very expensive, in particular when replicated for all sub-processes of the development process. E.g., what about design techniques? Do we need to perform a similar investigation? Actually, that is true: such analyses can turn out very expensive; on the other side, we believe that the testing example we reported about is one of the most expensive, because software literature is full of researchers and practitioners proposing their techniques and reporting their experiments in deeply different ways. For other parts of the development, e.g. for software inspections or safety cases, the literature can be much smaller and, consequently, cheaper to explore. Additionally, in general, it is not mandatory to make such studies for all possible activities: the construction of impact vectors can be incremental and new dimensions can progressively be added to the space whenever they become relevant. Of course, the sooner the dimensions are identified, the earlier corresponding data can be collected and used for process improvement. Finally, the analysis of literature to build baseline Impact Vectors is also optional: having such baseline enables leveraging a measurement-based approach to process enhancement since the beginning, because internal teams can leverage external teams’ experience. However, if such baseline does not exist, people can still process as they use to do in most contexts where data are not being used for process improvement or are not even collected. Overtime, data collected within the company will give birth to an internal repository of Impact Vectors. Such repository would probably have higher degree of internal consistency and will produce recommendations and results very reliable for internal teams; on the other side, it would probably be much smaller and with many more gaps with respect to a repository born from the aggregation of data coming the literature or, even better, from the collaboration of many companies sharing their data in a centralized repository (see Section VI, Software Engineering 2.0).

![Table Excerpt](image-url)

**Figure 44 Excerpt of the testing technique comparisons**

<table>
<thead>
<tr>
<th></th>
<th>functional (black-box) testing</th>
<th>control-flow testing</th>
<th>data-flow testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>boundary value analysis</td>
<td>sentence coverage</td>
<td>decision (branch) coverage</td>
</tr>
<tr>
<td>functional (black-box) testing</td>
<td>boundary value analysis</td>
<td>-0.300</td>
<td>-0.300</td>
</tr>
<tr>
<td>control-flow (white-box) testing</td>
<td>sentence coverage</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>condition coverage</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Chapter XVIII. **Sequencing Verification Techniques: an Experiment**

One of Squa’s believes is that changing the order of application of verification leads to drastically different results. However, accounting for all reasonable sequences of verification activities may turn very expensive. Therefore, before starting to collect data, Squa decides to make sure whether the order of verification techniques has some impact on the performance of the verification. Since software literature does not provide much information, he requires some form of experimentation.

In this chapter, we report on an experiment that investigates the performance of verification processes composed of the same verification techniques executed in different orders. Specifically, we report on two controlled experiments that Impact-One commissioned to us; the two experiments focus on sequences of two verification techniques: perspective-based inspection and unit testing. Results suggest that the performance of a verification process, in terms of recall and defect detection rate, strongly depends on the order of application of the techniques; therefore, interactions and synergies among verification techniques produce quality- and cost-relevant effects on the overall verification process that a decision maker should take into account.

**XVIII.i.1. Introduction**

Organizations aiming to develop highly reliable software systems, e.g. Defense or Space companies and agencies, clearly rely on a highly effective verification process and put a lot of effort in its definition. Also for organizations in other domains, where the reliability is desirable, an efficient and well-working verification process is recommended in order to maintain an agile and flexible process while assuring the adequate quality. Defining such a verification process includes decisions on the verification techniques to apply, on their order of execution and on the effort to allocate.

Several verification techniques have been extensively studied in literature and adopted in industry. Moreover, some combinations of more than one verification technique have been defined, investigated and applied, as described in more detail in the next Section. In this paper, we want to study sequences of verification techniques, as opposite to combinations [22], this last being a term that brings to the independence from the order of application [23]. Our hypothesis is that the order of execution of the verification techniques has a significant impact on the overall verification process performance. Consequently, decision concerning the verification process should be made while keeping into account synergies and mutual
interactions among verification techniques, so to maximize the benefits that each technique can get from techniques applied prior to it.

In the following paragraphs, we will report on our experimental investigation on two research questions. The first one is RQ.1: does the order of application of verification techniques impact on the overall verification process performance?

Furthermore, we know from our experience and from Impact-One qualitative analyses, that the definition of the verification process is commonly a continuous and iterative task that requires decision makers to define the next techniques to apply during the development; therefore, the verification process is not defined entirely and solely upfront. Such quasi-real-time decisions are made taking into account at least the following aspects: (i) the already achieved quality, (ii) the expected quality, and (iii) the available resources. Thus, it is useful to investigate and address the following further research question: (RQ.2) does the performance of a verification technique depend on the previously applied techniques? RQ.2 is fundamentally different from RQ.1. In fact, while RQ.1 addresses a planning problem (i.e. the definition of the most performing verification process for the current project), RQ.2 addresses a control problem: given the current state of the project, which verification technique(s) should be applied next in order to achieve the expected quality goals?

In order to answer the two previous research questions in a concrete scenario, we report on two controlled experiment taking into account two verification techniques: Unit Testing [22] and Perspective-Based Inspection [24, 25]. Because of the limited available experiment subjects, we narrowed our study to one type of artifact, i.e. Java code, and to the aforementioned verification techniques.

As we will be explaining in the next paragraphs, results show the importance of accounting for the order of application of verification techniques, as different orders lead to statistically different results. Therefore, one of our conclusions is that a strategic approach in defining the verification process is highly recommended, e.g. the one we proposed in [26, 27], because it would focus the problem of ordering the techniques rationally, and would allow leveraging the mutual influences between techniques. In particular, we will show how a strategic vision in defining the verification process can support decision makers addressing diverse practical problems efficiently, e.g.: given a predefined budget, which process maximizes the performance within that budget? Given a performance goal, which verification technique should be applied to minimize the effort required to achieve the goal?
XVIII.i.2. Related Work

A wide literature remarks the extensive importance for a well-defined verification process [28, 29, 30, 31, 32, 33, 34] and many empirical studies, comparing verification techniques to evaluate their performance, date back of many years [33, 34, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46]. Results reported by these papers are sometimes contradictory and report strongly different performances for the same techniques; such differences may derive from a myriad of causes and experimenters struggle to control factors that can potentially disturb their specific investigations. As we will be reporting, one of such factors can be the verification techniques executed upfront the investigations on the experiment objects.

Surveying the literature also revealed some papers addressing the problem of combining verification techniques. In particular, some of them assert that the overall verification performance may benefit from the combination of different verification techniques [22, 45], and some others propose and experimentally investigate concrete combinations [47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64], including the combination of inspection and testing [47, 56, 58, 62, 63, 64]. However, such studies mainly focus on the capability of different techniques to detect different defects; consequently, their combination results in a higher number of detected defects for reasons intrinsic to the techniques being used, i.e. their capability of targeting particular defect types, so that their union gives rise to a bigger set of found defects.

Some more studies go further and propose more elaborated combinations, consisting of a number of verification techniques to be applied in pre-planned sequences [44, 47, 59, 60]; the presence of such pre-planned sequences implies that the order of application may have a disruptive importance, generally because the output of a technique is required as input to the next technique; consequently, violating the order would turn the process unfeasible.

Moreover, slightly different approaches attempt to adjust and address testing by leveraging information gathered via inspections or other analysis techniques [47], which recall to some form of order on verification techniques; nevertheless, in these latest studies, techniques to be applied first are not used for detecting defects, but mainly for test case selection and prioritization. Additionally, in the past, an iterative approach for systematic selection of verification strategies was proposed in [49], where the authors also sketch the possibility that the order of application of verification techniques might influence the cost-effectiveness of a verification strategy, but they do not report any specific empirical study.

Based on this literature survey, the idea of sequencing treatments seems to be quite felt in the field of Software Engineering, and its importance has already been recognized in scientific
research also in other fields [65]; few examples of the so called “sequencing treatments” exist not only in Medicine [66], but also in Psychology [67], Psychiatry [68], and Human Sciences [69]. However, not much emphasis has been put on the experimentation and comparison of different and feasible sequences of verification techniques in Software Engineering, which is the goal of our investigation. In fact, in our literature survey, we could identify only one study doing so [64]. In that study, authors compare two different sequences of code inspection and unit testing. In our study, we propose again the comparison between sequences of the two same techniques, but we also introduce some changes in the experiment settings. The first difference is the reading technique – usage scenarios for [64], perspective-based for the present study. The second difference is on the time allocated for each task: unbalanced in [64] (2h30m for testing, 1h30m for inspection), balanced in our experiments (1h for each technique). A third and most important difference is that, in [64], the same team executes both techniques in a row, while, in our experiment, two different teams execute two distinct techniques (one each, in sequence) and the first team hands off the produced list of defects to the latest team. Further details motivating our design choices are provided in the next Sections, along with results of our investigation.

**XVIII.i.3. Experiment Description and Design**

*Experiment Overview*

The purpose of this study is to evaluate the effect that ordering verification techniques has on the verification process performance, from the point of view of a decision maker planning for or controlling the verification process.

The experiment factor to investigate is a two-step verification process \(<T_1, T_2>\), where the syntax \(<T_1, T_2>\) denotes a sequence of two verification techniques and means that the technique \(T_1\) is to be applied before the technique \(T_2\). Since we focus on Perspective-Based Inspection (PBI) and Unit Testing (UT) on code artifacts, we define the following treatments, each corresponding to a Process Performance Impact Vector (PPIV): \(<\text{PBI, UT}>, <\text{UT, UT}>, <\text{UT, PBI}>, <\text{PBI, PBI}>\). The main focus of the experiment is on comparing \(<\text{PBI, UT}>> against \(<\text{UT, PBI}>\), which will address RQ.1. However, in order to address RQ.2, we made the decision to include \(<\text{UT, UT}>> and \(<\text{PBI, PBI}>> as a way to assess the performance of the single techniques within a treatment.

In order to have comparable results between UT and PBI (i.e. to have a fair comparison), UT is intended to include defect identification, i.e., defect detection and subsequent defect localization in the code. This way, PBI and UT produce the same type of information [37], i.e. they detect the presence of a defect and its location in the code.
The maximum time allocated to each technique is one hour; the duration of the stages is balanced and, consequently, the treatment time is fixed to a maximum of 2 hours. Finally, we planned to execute the experiment twice, the second time as a replica of the first time with different subjects, in order to reinforce the study validity.

**Metrics**

The measures of performance we consider are: Recall (R), defect Detection rate (D), and percentage of False positives (F).

R is defined as the ratio of found defects to the number of known defects in the artifact. It represents a way of measuring the effectiveness of the technique.

D is intended as the ratio of found defects to the time spent for finding them. It adds cost considerations to the effectiveness and we use it as a way of measuring the efficiency of the technique.

F is the ratio of number of false positives to the number of report records (i.e. the one-complement of precision); a false positive is a detection which does not elicit a defect. This last metric is relevant because each false positive, as well as each defect, provokes some rework to be performed in the future; however, rework caused by a false positive should be minimized, as it does not produce beneficial effect. Whether a report record is a defect or a false positive is a decision of a software engineer not otherwise involved in the experiment.

These 3 metrics are obviously three dimensions of the impact vectors.

**Context Selection and Defect Seeding**

Subjects are 18 graduate students with experience with small- and medium-sized projects. They took classes in Software Engineering, Requirement Engineering, Object-Oriented Analysis, Design and Programming, and Software Architecture. Subjects also received specific training on the testing and inspection fundamentals and procedures to be used in the experiment. Subjects involved in each experiment were 9. The rationale for selecting such subjects is the small cost and the availability; in fact, the experiments run in the academic context, during class hours of the Experimental Software Engineering course, a teaching of the second year of the Master’s degree in Computer Science and Software Engineering at the University of Rome – Tor Vergata.

The experiment artifacts were portions of an academic project developed during a previous course in the first year of the same Master’s degree. The application is web-based, it is composed of around 5500 lines of Java code, and it manages university courses, student enrolments, exams, exam sessions and several user profiles. None of the experiment subjects worked on that project before. Different units of such project were used during each session of
the experiment. Objects were selected in order to achieve the maximum homogeneity in terms of complexity and size.

Considered structure, size and complexity of the code, our decision was to seed seven defects per artifact. The average size of code artifacts is 607 ELOC, thus the average defect density is 1.15% [70, 71, 72, 73], which should characterize the software quality of the experiment artifacts as poor [74]. Since we wanted to reproduce the first verification task in the development process (i.e. the first verification technique applied right after writing the code), this density seems to us both realistic and adequate for not getting the subjects exposed to too high a number of defects, which could cause the subjects to get lost in not-working software.

The defect seeding procedure consisted in introducing (i) initialization defects and (ii) mathematical and logical mutants [75].

**Experiment Design**

The four treatments (i.e. <UT, PBI>, <PBI, UT>, <PBI, PBI>, and <UT, UT>) were administered by providing subjects with two techniques in two sequential stages, whose duration is up to one-hour each. Teams were allowed to complete earlier than one hour, in case all team members agreed upon considering the task completed. This way, we wanted to measure the efficiency of each technique based on the actual time spent to accomplish the task.

The identification of the detected defects was requested, but not their correction; consequently, the output of each treatment provided the same amount of information, that is: (i) the list of defects detected with respect to the given requirements specifications, (ii) the duration of the executed technique, and, (iii) for each defect, the line(s) of code where the defect was located.

For each subject, a use case description and a sequence diagram were provided as specification of the functionality to examine, in addition to the related Java code. When performing UT, subjects were instructed to create a set of test cases and write some lines of code (i.e. test drivers) for testing the functionality. Also, there were expected, and enforced by the observers, not to read the code before writing all test cases.

When performing PBI, instead, subjects were expected to read the code, focusing on the assigned perspective, a non- compilable version of the code was provided, so to disallow them to execute the unit. Based on the artifacts to be inspected, the following two perspectives were defined and assigned: (A) Integration perspective, which takes into consideration: parameter passing, validity of input parameters, compatibility of interfaces, and compatibility and use of output parameters; (B) Unit perspective, which takes into consideration: logical correctness of
functions (i.e., in object-oriented terms, classes and bodies of their methods), correct elaboration of input parameters, and correct generation of outputs.

Because of the limited number of subjects, we had to limit the number of perspectives to two. The assignment of the two perspectives is random within the team, with the constraint that each of the two perspectives is implemented the same number of times by each team member during the experiment. This way, we plan to mitigate the influence of each subject’s skills on the experiment outcome.

Subjects performing PBI worked in a team composed of two people, one perspective per member, while teams performing UT were composed of one subject. Teams were not allowed to directly interact and communicate at any time.

When the treatment is <PBI, PBI > or <UT, UT>, the same team works on the same artifact for both stages. When the treatment is < PBI, UT > or <UT, PBI >, the team assigned to the first stage is different from the team assigned to the second stage. This last will receive the list of defects produced by the first team; the list is anonymous, so there is no way for the second team to find out (and possibly interact) with the first team.

Figure 45 and Figure 46 provide a graphical explanation of the experiment design. In the upper part of Figure 46, the same team executes twice UT, whilst, in the lower part, two different teams execute the two different techniques.

The motivation for such design is that, in a real context, in particular for systems where verification is required to be very accurate (e.g., Space and Defense systems), it is frequent and reasonable to expect different teams to perform code inspections and unit tests. In our study, we wanted to reproduce such a context.
A final remark on the design is that no specific guidelines were provided on how each team was expected to use the list of defects received by the previous team; therefore, the team executing the second technique of the process was free to evaluate each record of the received list and establish whether it was a defect or not. In case it was, the team reported that record in its own report; otherwise, the record was discarded and was not part of the verification process output.

Subjects used their own laptops, where they had previously installed Eclipse Helios [76], and Java EE 6 [77], with which all of them were familiar.

**Mitigation of the Impact of Subjects’ Skills on the Experiment**

Before each experiment, we ran a pilot study to assess the capability of the subjects at identifying defects with PBI and UT. Based on the results of such pilot studies, we created three groups of three subjects for each experiment; the rationale for assembling groups was to minimize the variance among group capabilities, so to minimize impact of subjects’ individual capability on the experiment results.

The two subjects constituting each PBI team are picked up from the same group, while the third subject of the group performs UT on the same artifact; in the next step, the subject who performed UT performs PBI, together with one of the two subjects previously involved in PBI, on a different artifact. Then, the subjects who performed twice PBI performs UT, while the remaining two subjects perform PBI on fresh artifacts. This way, after three rounds, there will be 3 artifacts, as exemplified by Figure 46, on which the group (as a whole) has performed both PBI and UT (with different subjects) and, thus, has produced the list of detected defects. Such artifacts and the corresponding identified defects are then provided as input for the teams that perform the second techniques of the treatments. In particular, as aforementioned and represented in Figure 46, if the randomly selected artifact for the treatment <PBI, PBI> is A1, than subjects 1 and 2 will perform one more hour of PBI on A1; if A2 is the artifact selected for <UT, PBI>, then it will be delivered to a subject not belonging to the same group of subject 2, together with the list of defects produced by subjects.

**Design Validity**

To be as close as possible to a real environment, during the experiment design, we paid attention not to have the same team work on the same artifacts with both techniques (PBI and UT). In general, in fact, UT may be not performed (and, in our reference customers, usually it is not) by the same team performing PBI and vice versa. Letting the same team perform both UT and PBI on the same object could have biased the experiment results: in such a case, in fact, the performance of the second technique of the treatment could benefit from the knowledge
gained during the execution of the first technique, which would cause the performance of the second technique to be biased.

The participation in the experiment was optional and the subjects were not evaluated based on the experiment results. To make the effect of maturation over all treatments homogeneous, the order in which treatments were executed was balanced across time (i.e. each treatment had to be applied at least once before any treatment being applied for the second time). Therefore, randomization was used, but in respect of such constraint.

Generalization of results to an industrial context is limited due to the fact that (a) graduate students rather than professionals participated in the study, and (b) experiment objects were different portions of the same software project.

Since the goal of this study is not to identify the best verification process, but rather to provide evidence on the importance of a correct and well-reasoned sequencing of verification techniques in each context, we consider acceptable such validity issues.

**XVIII.i.4. Result Analysis**

The first question to answer is RQ.1: does the order of application of verification techniques impact on the verification strategy performance? In order to answer, we need to compare the performance of the two treatments \(<PBI, UT>\) and \(<UT, PBI>\). First we run normality tests: p-values of the Shapiro-Wilk test are reported in Table I for both experiments; the tests show that most samples are normally distributed, except in 3 cases. The chosen alpha is 0.025 and significant p-values are bolded and underlined.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Treatment</th>
<th>p-values of the Shapiro-Wilk tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall (R)</td>
<td>Defect Detection Rate (D)</td>
</tr>
<tr>
<td>1</td>
<td>(&lt;PBI, UT&gt;)</td>
<td>0.756</td>
</tr>
<tr>
<td></td>
<td>(&lt;UT, PBI&gt;)</td>
<td>0.458</td>
</tr>
<tr>
<td>2</td>
<td>(&lt;PBI, UT&gt;)</td>
<td>\textbf{0.018}</td>
</tr>
<tr>
<td></td>
<td>(&lt;UT, PBI&gt;)</td>
<td>0.726</td>
</tr>
</tbody>
</table>

Samples are independent by experiment design (different artifacts were used and teams were not allowed to directly interact); therefore we can run the independent t-test (parametric) and the Mann-Whitney test (non-parametric), according to normality of samples (see Table I). The conservative Bonferroni correction is used for assessing the significance and results of the
comparisons are reported in Table II and show that a statistically significant difference exits in terms of R and D, but not in terms of F.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>p-values of the comparisons (independent t-test or Mann-Whitney test)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall (R)</td>
</tr>
<tr>
<td>1</td>
<td>0.007</td>
</tr>
<tr>
<td>2</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

For completeness, we computed the power of the test: in the first experiment, the means are 0.679 and 0.520 for R (<PBI, UT> and <UT, PBI>, respectively), and 2.107 and 3.236 for D; the common standard deviations are 0.111 and 0.618, alpha is 0.025 and each sample size is 9. Therefore, the test powers are 0.79 for R and 0.95 for D. In the second experiment, the means are 3.320 and 1.636 (<PBI, UT> and <UT, PBI>, respectively); the common standard deviation is 0.678, alpha is 0.025 and each sample size is 9. Therefore, the resulting test power is about 1 in both cases.

The second question to answer is RQ.2: does the performance of a verification technique depend on the previously applied techniques?

In order to answer, we need to verify if any differences exist in the performance of PBI (or UT) when its history changes (i.e. different techniques are applied prior to it). If performance does change, then we have evidence that the performance of PBI (or UT) can depend at least on the previously applied technique (i.e. PBI, UT or none).

We want to investigate the following comparisons, where the asterisks (*) indicate the techniques whose performance are being compared: <PBI*, ?> vs <PBI, PBI*> vs <UT, PBI*> and <UT*, ?> vs <UT, UT*> vs <PBI, UT*>.

In order to compare the results, we need to define R and D for a single technique. R is computed as follow: if the technique is the first of the strategy, then R is defined as the ratio of found defects to the number of known defects. If the technique is the second of the strategy, then R is defined as the ratio of new defects found during the second stage to the difference between known defects and defects detected during the first stage. Accordingly, D is computed as follow: if the technique is the first of the strategy, then D is defined as the ratio of found defects to the hour of effort required by the technique. If the technique is the second of the strategy, then D is defined as the ratio of new defects detected during the second stage to the effort required by the technique.
The first test to run is to check the homoscedasticity of samples, which would enable the use of One-Way AnOVA. In the first experiment, significances of the Levene’s test for R, D and F are .064, .360 and .000, while in the second experiment they are .243, .309 and .004, i.e. only the variances of F are homogeneous. Therefore, AnOVA can be used for the comparison of F samples, but not for R and D, where Kruskal-Wallis can be used. When a difference is detected by AnOVA or Kruskal-Wallis tests, additional investigation is useful: in fact, in order to understand which pairs of treatments are different in terms of R and D, Mann-Whitney tests can be performed. Similarly, a post-hoc analysis (Bonferroni test) can be used on F samples, when the AnOVA test detects a statistically significant difference. P-values of such pairwise comparisons are reported in Table III, Table IV, Table V and Table VI; cells with a dash (-) indicate that no statistically significant difference has been detected by AnOVA or Kruskal-Wallis (therefore no pairwise comparison is needed); symbols “>” and “<” indicate the direction of the pairwise disequality. Alpha is always 0.025 and significant p-values are bolded and underlined.

**TABLE III. PERFORMANCE OF PBI: 1st EXPERIMENT**

<table>
<thead>
<tr>
<th>P-values of the comparisons and direction</th>
<th>Recall (R)</th>
<th>Defect Detection Rate (D)</th>
<th>Percentage of False Positives (F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;PBI*, ?&gt; vs &lt;PBI, PBI*&gt; vs &lt;UT, PBI*&gt;</td>
<td>-</td>
<td>-</td>
<td>&lt;0.001 (&lt;)</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>&lt;0.001 (&lt;)</td>
<td>0.008</td>
</tr>
<tr>
<td>&lt;PBI*, ?&gt; vs &lt;PBI, PBI*&gt; vs &lt;UT, PBI*&gt;</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**TABLE IV. PERFORMANCE OF PBI: 2nd EXPERIMENT**

<table>
<thead>
<tr>
<th>P-values of the comparisons and direction</th>
<th>Recall (R)</th>
<th>Defect Detection Rate (D)</th>
<th>Percentage of False Positives (F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;PBI*, ?&gt; vs &lt;PBI, PBI*&gt; vs &lt;UT, PBI*&gt;</td>
<td>-</td>
<td>-</td>
<td>&lt;0.001 (&lt;)</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>&lt;0.001 (&lt;)</td>
<td>0.008</td>
</tr>
<tr>
<td>&lt;PBI*, ?&gt; vs &lt;PBI, PBI*&gt; vs &lt;UT, PBI*&gt;</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**TABLE V. PERFORMANCE OF UT: 1st EXPERIMENT**

<table>
<thead>
<tr>
<th>P-values of the comparisons and direction</th>
<th>Recall (R)</th>
<th>Defect Detection Rate (D)</th>
<th>Percentage of False Positives (F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;UT*, ?&gt; vs &lt;UT, UT*&gt; vs &lt;PBI, UT*&gt;</td>
<td>&lt;0.001 (&gt;)</td>
<td>&lt;0.001 (&lt;)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>&lt;0.001 (&lt;)</td>
<td>&lt;0.001 (&lt;)</td>
<td>-</td>
</tr>
<tr>
<td>Comparison</td>
<td>Recall (R)</td>
<td>Defect Detection Rate (D)</td>
<td>Percentage of False Positives (F)</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>---------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>&lt;UT, UT*&gt; vs &lt;PBI, UT*&gt;</td>
<td>0.019</td>
<td>0.050</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table VI. Performance of UT: 2nd Experiment**

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Recall (R)</th>
<th>Defect Detection Rate (D)</th>
<th>Percentage of False Positives (F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;UT*, ?&gt; vs &lt;UT, UT*&gt;</td>
<td>-</td>
<td>0.049</td>
<td>-</td>
</tr>
<tr>
<td>&lt;UT*, ?&gt; vs &lt;PBI, UT*&gt;</td>
<td>-</td>
<td>&lt;0.001 (&lt;)</td>
<td>-</td>
</tr>
<tr>
<td>&lt;UT, UT*&gt; vs &lt;PBI, UT*&gt;</td>
<td>-</td>
<td>0.001 (&lt;)</td>
<td>-</td>
</tr>
</tbody>
</table>

**XVIII.i.5. Results Interpretation**

Based on the reported results, we can detect significant differences both in terms of R and D when switching the order of execution of PBI and UT. This means that, in the context under study, a verification process strongly feels the effect of changing the order of application of techniques.

Given such considerations, we can start reasoning on a number of practical problems, e.g. given a predefined budget, which verification process maximizes the expected performance? As shown by our statistical analysis, different conclusions can be drawn, depending on the context the verification process is being planned for. If the goal of the verification process is to maximize the percentage of detected defects with respect to the existing defects (i.e. to minimize the number of remaining defects), then the verification planner should prefer <PBI, UT> to <UT, PBI>, as it...
has the highest recall (see Figure 47, plot a and plot b). This can be the case of safety-critical systems, where the goal is generally to remove as many defects as possible. Nevertheless, in different contexts, cost-effectiveness aspects can play a major role and the defect detection rate could be the primary criterion to select the verification process; in such contexts, based on our results, the decision would be uncertain, because results of the two experiments for D are contradictory (plot c vs plot d). However, as already mentioned, the goal of this study is not to conclusively identify the best sequence of UT and PBI on code artifacts, because at least larger samples, professionals with diverse expertise, and a real-world-project would be required for such a conclusion. Conversely, this part of the study wants to show the importance of the order of verification techniques constituting a verification process, as it can make a choice to prevail over another.

In conclusion, the answer to the first research question is that, based on our results, the order of application of verification technique can have a remarkable impact on the verification process performance, and therefore it should be carefully considered when planning a verification process.

Concerning RQ.2, results reveal that very often the performance of a verification technique changes when the history of the verification changes. In both experiments, it results that the defect detection rate of a technique (either PBI or UT) varies when it is applied as first or second technique of the verification process: this is true in 8 cases out of 12 comparisons. However, variations are less frequent in terms of R (2 times out of 12) and F (3 times out of 12).

In brief, we can say that, for the context under study, the efficiency (i.e. D) of unit testing and perspective-based inspection depends on which techniques were applied before them; conversely, the overall recall of the verification process, as well as percentage of false positives, tend to be unchanged most of the times.

**XVIII.i.6. Conclusion**

In this paragraph, we addressed two research questions related to the definition of a verification strategy: (1) does the order of application of verification techniques impact on the verification strategy performance? (2) Does the performance of a verification technique depend on the previously applied techniques?

Two controlled experiments were performed to investigate such questions. Synthesizing the obtained results, we can affirm that the verification process performance depends on the sequence in which verification techniques are applied and, more in detail, the performance of each technique can change based on the history of the verification process. Results supporting these conclusions bring up one more aspect of complexity to the attention of a verification
planner. In fact, they show that, in order to undertake fully informed decisions, a verification planner has to include in his/her reasoning considerations on the order in which verification techniques will be applied; in the act of controlling an on-going process, furthermore, the verification planner should then make decisions also based on which techniques were already applied and should use historical measures carefully: any given verification technique may perform differently from the past, if applied at a different stage of the verification process, if the current verification process is a different sequence of verification techniques with respect to the past.

Results should encourage Squa and Impact-One to further investigate how to correctly sequence verification techniques, in order to empirically elicit functions which model the mutual influences among techniques (i.e. their combination functions) and, consequently, their expected performance (i.e. their activity performance functions).

Chapter XIX. Test Performance Evaluation

A couple years are gone since the company led off the new software group, along with the innovative approach of impact vectors. Management has invested a lot of resources in the new software intergroup and it is time to start evaluating how procedures and heuristic put in place are behaving. In this chapter, we focus on the investigation of how test is performing with respect to detected defect types. In particular, Impact-One has created repository of 3179 bugs detected by a slice of the software group across 10 projects; this slice is composed of: the architect, an analyst, a designer, and three programmers and they form a team working with web technologies, i.e., within the company, the server/client components enabling physicians to access the repository that contains all patients data, and which supports them at making their diagnoses.

If we focus on the bug repository, we can find 53 impact vector dimensions that characterize a bug, among which we want to mention the following: Status, Priority, Original Effort Estimate, Actual Time Spent, Security Threat, Defect Category, Defect Severity, Phase of Detection, Iteration Number within the phase, and Tool/Method Used for detection.

In the rest of this chapter, we will report on the most relevant and interesting results of the analysis of the impact vectors.
XIX.i. Analysis of Results

The first dimension that John Soft (Software Development Lead) and Squa (Software QA lead) want to investigate is the defect type. Upfront, the software division identified a taxonomy of 14 defect types:

- Requirements defects:
  - Misunderstood requirement
  - Missed requirements

- Code defects:
  - Logic
  - Formatting
  - Content
  - HTML
  - Data
    ı. Access
    ıı. Migration (between servers)

- Security defects
- Configuration defects
- Supplier defects
- Usability defects
- Accessibility defects
- Non-standard compliance

Figure 48 Distribution of bugs across defect type
Classification is orthogonal. As evident from Figure 48, most of the defects are code-related, in particular on the code logic. Since the web application consists of a big amount of GUI code, it is not a surprise that many bugs fall in user interface-related categories, e.g. formatting or HTML. However, the distribution of defect types does not convince the QA team, so they decide to investigate some groups of related defects across development phases. Concerning the web development, we did not describe in detail the implemented process. Since we are focusing only on test performance, the following process information is relevant: unit test is performed during the development (first phase). After development, during system integration (second phase), Internal System Testing (IST) is executed. Eventually, the validation of the system happens via User Acceptance Testing (UAT), where the client participates in the test activity with its perspective. No other test activities are performed on this category of software (web software) within the company, so defects can either be detected during development, or during IST, or during UAT or, worst case, when the software is in production.

The data we present in the following are about projects that are not already dismissed, but delivered to the customers in the form of stable versions; therefore, even though the number of bugs is not conclusive, it is expected to be very close to the real number of bugs that will be detected during the entire life cycles of those projects.

In terms of Impact Vectors, therefore, we have 4 Process Performance Impact Vectors, i.e. one per development phase and each representing a sub-process of the software development process enacted by the teams. These sub-processes are composed of several activities, including verification activities that, in the specific taxonomy of the involved teams, are called “Detection Method”, which is, as we mentioned above, one of the dimensions in the repository available to us; the nominal scale for such dimension is composed of the following members: Manual Testing, Peer Review, Product Usage, UI Testing Tool, and Usability Testing. For each activity execution (i.e. the application of any of these verification activities) some bugs are detected and reported. This way, the repository is a collection of bugs characterized by the detection method that found it (e.g. usability testing), the defect type (e.g. code logic), the detection phase (e.g. production) and so on. From this repository, we can build the impact vectors corresponding to each verification activity and to each development phases. Such impact vectors for verification activities (i.e. Activity Performance Vectors) and process phases (i.e. Process Performance Impact Vectors) are composed of dimensions of interest to management, including: number of detected defects for each defect type; number of actual defects known as of today; effort spent to run the activity/phase; distribution of the severity of defects detected by the activity. By having such impact vectors, a number of interesting analyses is enabled and facilitated, which can lead to actionable feedbacks and interesting
insights. Some of such analyses, performed by leveraging the part of the tool chain presented in the previous Section, are reported below, starting by exploring the defects by detection phase. If we look at the distribution of defect across their corresponding detection phase, we see something not really pleasant for quality assurance people: defects seem to be detected late in the development. In fact, 22% of defects are shipped to the customer (in production), 30% is found during user acceptance testing, 31% is found during system internal testing and only 7.5% during development (see Figure 50).

One first guess was that defects slipping through were not really relevant, but just minor bugs reported by picky developers. Actually, this hypothesis can be discarded when looking at bug severity, as reported in Figure 51: fatal and high severity bugs are frequent and around 50% of them is at least medium, i.e. not something the team can forget about.
If we proceed with the analysis by defect type, we detect that different trends with respect to the global trend.

Concerning requirements defects, there is a slippage of 54% from the IST phase (59/110 bugs, i.e. 38 to UAT and 21 to production), and a slippage of 89.09% (98/110 bugs) during development: 9 out of 10 defects are not caught during development.

When we advance to accessibility and usability defects, the trend seems similar: most of the defects are detected late in development.
However, while the trend is similar, its interpretation is deeply different. Requirements defects can and must be detected early in the development and should not slip through. On the other side, many usability defects cannot effectively and efficiently be removed upfront, whilst accessibility defects, at a careful analysis, resulted cheap to address even when discovered in UAT or even in production. Therefore, we can live with a late detection of usability and accessibility defects, but we should definitely enact some countermeasures to target requirements defects.

Next step is to investigate data-related defects.

Also in this case, as for the requirements defects, late discovery means much higher costs of removal and a slippage of 73% (201/275 bugs) is something to target with particular care. More specifically, SQUA should identify testing techniques, both at unit and integration level, to detect data-related bugs earlier.

For what concerns GUI-related defects (i.e. HTML, content and formatting), one should not expect to detect many of them during the early phases, as they are refined and tuned mostly during integration, when complex web flows are composed to implement complete use cases. The trend in Figure 55 is quite expectable, even though the number of defects detected in UAT and, in particular, in production, is still high. At a closer analysis, about 90% of the defects detected in the last two phases requires less than 1 hour to be fixed; this means that the vast majority of issue reported at that time consists of tweaks, for which reasonable and convenient techniques for early detection are not known to us.
Last slice of analysis is on code defects, in particular logic defects, which represent a significant part of all defects. Data are pretty clear: 91% of the bugs (1235/1358) are slipping through the development phase and 60% (811/1358) is even slipping through IST. After this warning was raised, management started to interview developers and to explore a significant sample of the defects to better understand their real nature. Unfortunately, even though some misclassifications exist, most of the defects are correctly classified, which means that quality control (and in particular unit testing) is not really working efficiently for web projects. On the other side, however, the mentioned investigation also revealed that many logic defects detected during development are not being recorded. This is not necessary something bad: recording defects has a cost and, if such knowledge does not produce useful information, then collecting data becomes a waste. Also, the investigation revealed that many logic defects are corrected in just a few minutes and only about one third requires more than one hour.

Based on this, on one side, instructions were given on how and when record bugs detected during development. In short, fast-to-solve bugs (<0.5h) can have a fast-track, where just a couple of fields need to be filled in the bug report form; the other bugs must be recorded in full. The goal of this new procedure is produce more realistic trends. On the other side, the inefficiency of unit testing is still a fact, because rebalancing the proportions across detection phases does not solve the issue of having a big (absolute) number of defects being shipped (or at least exposed during UAT) to the customer. Therefore, results reported in chapter XVII were leveraged in order to prepare a series of training sessions about unit testing techniques and approaches.
Concerning an effort analysis, only 662 records (20% of total) report effort information; in particular, data were collected only during some projects. This is something negative; in fact, in our context, one of the most meaningful ways of measuring the efficiency of a V&V technique is the defect detection rate: \#bugs found / effort spent. However, no information about detection effort, localization effort and correction effort are being collected and distinguished. As an example, consider manual testing (the most used technique); via manual testing, 400 bugs were detected and 517 person-hours (ph) were logged for those bugs. So, one could say that the efficiency for manual testing is 400 bugs divided by 517ph, i.e. 0.77 bugs/ph. Nevertheless, correction time for those bugs is contained in those 517ph, but such effort should not be accounted for computing the technique efficiency. For instance, how much effort did the team spend on manual testing without finding any bugs? That effort should be accounted when computing the efficiency of manual testing, as it is part of the effort allocated to that technique, even though unproductive.

Based on these results, management decided to organize some training courses concerning testing techniques capable to target data-related defects. Also, they decided to encourage a finer and more careful data collection in terms of effort, because it represents one of the most important decision drivers. Finally, in order to address requirements issues, the management decided to hire a new requirements specialist and to acquire both a requirements management tool and a support tool for specific for requirements reviews.

In conclusion, using data collected overtime enabled the construction of impact vectors, whose main dimensions consists of those attributes that enable modeling the capability of each detection technique an process phase to find defects of certain defect types. Such modeling of the process supports an easy and quick derivation of recommendations on how to target...
inefficiencies of the process and identify resolution strategies. Such resolution strategies were not direct outputs of the impact vector model; in fact, even though we would like our framework to provide at least some alternatives to the decision makers, we have not achieved that level of advancement for our framework. Nevertheless, the impact vector framework is already able to provide a clear, significant and useful bunch of knowledge that elicits punctual issues in the development process, so that decision makers can be supported to actually and concretely identify appropriate solutions to these precisely identified process issues.

Chapter XX. Selecting Technologies for Increasing Productivity: an Example with Mobile Technologies

XX.i. Introduction

The company decided to explore a new field, i.e. the mobile application development. In fact, the company is opening a new branch in its market to target self-monitoring device, that is device that transmit data to a mobile phone, where an app is installed which elaborates those data and presents them to the user in an understandable way (i.e. alerts); apps can also be distributed to other members of the family, possibly assigning limited access rights, and to doctors in order to augment their buzzers and provide quasi-real-time details on the health condition of patients requiring particular care.

The general question that this paragraph tries to answer is: what is the right technology to use for the development of a mobile application for given requirements and context? So the company performed a broad study to explore the most popular and advanced mobile technologies, in order to characterize them and be able to select the most appropriate, according to specific needs.

In order to develop a new Mobile App, in fact, several decisions have to be made. The first one is the platform on which the application will run. Such decision is usually made by the marketing management. Very often, more than one platform is chosen, although versions for the different platforms can be released at different times. Instances of platforms include, but are not limited to:

- iOS platforms,
- Android platforms.

As soon as the platforms decision is undertaken, one more very important choice is required, which matches the aforementioned question: What technologies should be used to develop the given mobile app? Now, the Technical Leader has the ownership of such decision. As we will be explaining in the next Sections, the most known examples of such technologies include:
• HTML5;
• CSS;
• JavaScript;
• Platform Specific Software Development Kit (SDK), including:
  ○ Android SDK, and
  ○ Apple® XCode;
• Titanium Mobile, and related supports for Mobile App development;
• Adobe®, and related supports for Mobile App development;
• PhoneGap;
• jQueryMobile;
• Sencha Touch;
• DojoMobile.

Indeed, even when a single platform is initially targeted to develop an app, few ways exist to enact the development, which can be classified in three Development Approaches (DAs):

- Native DA: i.e., pure native development, using the platform specific SDK;
- Web DA: i.e., pure web app (HTML 5 allows Apps that are almost as powerful as native apps);
- Hybrid DA: i.e., using a mixed approach, as supported by several different technologies discussed in the remaining of this chapter.

A decision concerning the DA to use has to take into account both functional and non-functional app requirements, such as the need for different devices on board (GPS, Camera, etc), as well as constraints placed by the selected target platforms, schedule, and budget. Therefore, a multi-criteria decision process is required and both tools and frameworks would desirable to support and guide it.

XX.ii. Purposes of the Study and Research Methods

Evidence-based research in the field of software development for mobile apps is in its beginning and still needs enhancement. Data and knowledge bases are not yet publicly available. Concerning private companies, it may be that data collections are not yet capitalized organization-wide. As a consequence, it is still frequent for researchers to start studies by producing their own data from scratch.

The research methods that we used for this study are survey and case study research. For data collection, we used literature and technical documentation survey, interviews to practitioners, and development of case studies by using and comparing different technologies.

The purpose of this work is exploratory rather than confirmatory [78]; in fact, we do not aim to give a final answer to the basic research question, but we want to check whether an approach is feasible and capable to support answering the basic question. Consequently, in the
remaining, no formal goals [79], hypotheses [80] or details concerning the utilized research objects and processes (case studies, handbooks, papers, etc.) will be presented.

In general, we do not aim to address the false problem of defining the “absolute best” technology for mobile software development. Vice versa, we remark that our expectation is that no single development approach will fit all needs of all customers. As Fling observed [86], whatever medium we use to address our goals, we have a number of choices, each with its own pros and cons. Similarly to all other branches of Software Engineering, we expect some mobile media to be quick to create apps, but accessible to a few of the customers and/or delivering hard to maintain apps; others could address a larger market, but far more expensive and complex to use.

However, it is clear that companies are facing an obvious trade-off between user experience and application functionality on the one hand, and development costs and time to market on the other hand. Therefore, the challenge is to choose a development approach and a technology that are capable to balance requirements with the available budget and time-to-market [81]. At this point, the fundamental question is: “What is the right choice?”.

In order to help answer such question, this paper aims to provide

(i) a guide to technology decision, and

(ii) a framework to support such decision, which could evolve based on new technologies that will come out.

**XX.iii. Characterizing the Technologies to Study**

The choice of the platform dictates the range of technologies to use for developing an app. If we need an application expected to be multi-platform, then we have a range of technologies and possible approaches to use - besides developing the same app for each platform using their native SDK, - i.e. hybrid or web approaches. Conversely, if we need an app for a single Operating System (OS), then we have different choices available and the right approach could be the native development.

In the remaining, we will try to answer the initial question about mobile technologies by focusing on three Platform Categories (PCs):

- **PC1.** iOS platforms;
- **PC2.** Android platforms,
- **PC3.** Other platforms, which we merge under this category, namely “Others”.

So, it is fairly intuitive that the platform will be one of the impact vector dimensions.
As already mentioned, after the selection of the target platform(s), the next step consists in choosing both the Development Approach, and the Development Technology (DT).

There are three DAs available for apps development:

- **DA1.** Web apps
- **DA2.** Native apps
- **DA3.** Hybrid apps

Since they are mutually exclusive, one dimension is enough to model this aspect via impact vectors.

Concerning the DT, it is not independent from the selected DA and PC. In this chapter, we take into consideration a limited set of development technologies, among the ones available on the market. Specifically, we consider those technologies that the largest software companies commonly adopt worldwide. We have already listed such technologies in the previous paragraph. In the following, we consider these technologies and group them by the selected DA.

**Web apps:** choosing the technology is quite straightforward:

- **DT1.** HTML5, CSS, and JavaScript. Since these technologies fit together, we consider this set as a single choice.
- **DT2.** Native apps: the platform specific development kit can be selected, e.g.:
  - Android SDK;
  - Apple XCode.

  Additionally, the following technologies can support the development of Native Mobile Apps:
- **DT3.** Appcelerator®’s Titanium Mobile,
- **DT4.** Adobe® Flex, in conjunction with Adobe® Air.

  Last two development technologies can create cross-platform apps from the same source code.

**Hybrid apps:** PhoneGap can be used, in addition to the following JavaScript frameworks:

- **DT5.** PhoneGap + jQuery Mobile;
- **DT6.** PhoneGap + Sencha Touch;
- **DT7.** PhoneGap + Dojo Mobile.

In this case, we need to create one Boolean dimension for each technology; this way, each technological stack can be represented via an impact vector by means of a bit mask, i.e. by setting to true those technologies that are part of the technological stack and to false the ones not being part of the stack. In conclusions, the Boolean dimensions are: HTML5, CSS, plain JavaScript, Native, Appcelerator®’s Titanium Mobile, Adobe® Flex, Adobe® Air, PhoneGap, jQuery Mobile, Sencha Touch, and Dojo Mobile.
A brief review of the listed technologies, including their pros and cons, is provided in the remaining of this Section.

**XX.iii.1.  HTML5 + CSS3 + JavaScript (DT1)**

Since they are web technologies, the learning curve is expected to be very steep (i.e. we expect developers to quickly get familiar with them), if compared to the native languages. Additionally, when writing an application, multiple platforms can be targeted without encountering any significant problems. On the other hand, since HTML5 and JavaScript are not allowed to access all device features, the developer could encounter some limitations. Moreover, as an application built with such stack has to be interpreted by the browser, performance can be worse than an application developed with native languages.

**XX.iii.2.  Platform Specific Software Development Kit (DT2,3)**

It ensures the full exploitation of all features of the platform on which the practitioner decides to develop the app, in addition to the native look & feel. The most obvious downside is the ability to address only one platform; for this reason, if the organization is willing to produce a multiplatform application, the code has to be rewritten for each chosen platform. This contributes to get high development cost and long developing time. It’s important to underline that in the following the DT2 and DT3 will be considered together.

**XX.iii.3.  Appcelerator® Titanium Mobile (1.8.1) (DT4)**

Titanium Mobile is a cross-platform mobile development framework that enables developers to write JavaScript code, which is then compiled down to native code for the iOS and Android platforms. If the organization wants to build a multiplatform application, Titanium Mobile can save time with respect to developing via platform specific SDK. Additionally, a real native application is obtained, with the platform-specific look & feel.

**XX.iii.4.  Adobe® Flex Mobile (4.5) + Adobe Air (DT5)**

Flex is another cross-platform mobile SDK. It enables developers to write their applications by using the ActionScript language for Android, iOS, and Blackberry platforms. Concerning iOS, it generates a pure native application written in Objective C. Concerning Android and Blackberry, the developer has to install a further layer on which the application will run: the Adobe Air Environment. Applications developed by Flex are deployable on the most important app stores.

**XX.iii.5.  PhoneGap (1.7.0) (DT6, 7, 8)**

This technology enables developers to create a web application wrapped into a native container. This means that developers can write an application once, and, when they need to migrate to another platform, they can reuse the code by wrapping the initial web app into...
another native ad-hoc wrapper. Since both wrappers offer the same interface to developers, no changes are required to the previously written code. Developers still have to face the cons associated to using HTML5, CSS, and JavaScript (see Sub-Section A above), but at the same time they receive some of their benefits, including a steep learning curve, the possibility of deploying the app on appstores, and the chance of extending the native wrapper to use device features not available in the default HTML5.

Since PhoneGap allows to develop pure web apps, selecting a JavaScript framework appears to be an essential choice. In this paper, we take into account three main JavaScript frameworks, among the ones available on the market, i.e.: jQuery Mobile, Sencha Touch, and Dojo Mobile. jQuery Mobile seems to us to be a good candidate for very basic and logically simple applications, while Sencha Touch, due to the different development approach, has to be preferred when the organization is willing to leverage more complex development patterns, such as the Model-View-Control. It seems to us that this technology shows a very flat learning curve, but it gives practitioners the possibility to keep under management the development of even the more complex applications. Dojo Mobile seems to us to be the most complete technology, among the three considered above, as it provides both development approaches offered by jQuery and Sencha. However, if a simple business logic, small footprint, and very small time to market are needed, jQuery Mobile could be the right choice.

**XX.iv. Drivers for Technology Selection**

In our study, the technology selection is driven by a main set of requirements, grouped by the following two categories:

(i) Needs

(ii) Device features.

Based on literature [82, 83, 84, 85, 86, 87, 88, 89, 90, 91], we can breakdown these categories as in the followings. Needs represent both functional and non-functional requirements. Needs, or, equivalently, Non-Functional Drivers (NFDs) are:

NFD1. Access to native hardware features;
NFD2. High performance;
NFD3. Cross platform;
NFD4. Easily upgradeable;
NFD5. Deployable on app stores;
NFD6. Small footprint;
NFD7. Quick coding and prototyping;
NFD8. Complex
NFD9. Simple business logic;
Device Features (DFs), as commonly found on mobile devices, include:

- DF1. Accelerometer;
- DF2. Compass;
- DF3. Orientation;
- DF4. Light;
- DF5. Contacts;
- DF6. File;
- DF7. Geolocation;
- DF8. Media;
- DF9. Network;
- DF10. Notification Alert;
- DF11. Storage;
- DF12. Multitouch;
- DF13. SMS;
- DF14. Bluetooth;
- DF15. Video Capture.

Both NFDs and DFs are part of the impact vector dimensions.

XX.v. A Model for Evaluating Mobile Technology

Based on the content of previous paragraphs, answering to our basic question – What is the right technology to use for the development of a mobile application for given requirements and context? – means to solve a problem in a space made by forty dimensions: 3 related to PC, 3 to DA, 7 to DT, 12 to NFD, and 15 to DF. Three of these dimensions, the PC-related ones, are alternative dimensions; the same holds for the three DA-related dimensions. The twelve NFD-related dimensions are independent one another, and the same holds for the fifteen DF-related dimensions; however, DF dimensions can depend on NFD, DA, and PC dimensions, even though we might still be unable to express, a priori and formally, such dependencies. Additionally, an application is required to meet many needs, a device feature can serve many needs, and each development technology can offer different features, depending on the platform it is used for. In other words, we are not coping with a system that we can immediately and easily model via mathematical functions, assuming it is feasible.
To manage the complexity, our decision was to proceed by abstraction and classification, and we eventually consolidated those forty dimensions in five macro-dimensions. This led to constructing a five-macro-dimension space of: Needs, Device Features, Development Technologies, Platforms, and Development Approaches. The domain of each macro-dimension includes the items listed in the previous Sections for PC, DA, DT, NFR, and DF. Since we see no interest in sorting those items, in their domains, they are listed in arbitrary order, so obtaining a Nominal scale [80, 92] for each dimension.

This discrete space of five macro-dimension can be easily and automatically generated and updated whenever new platforms or development technologies/approaches appear on the market or are removed, for any reasons, from the set of accepted candidate technologies/approaches.

Our next step is to represent: (i) each technological stack as a point in this space, according to its capability to meet some requirements (e.g. support for a given device); and (ii) the app to develop as a point in the same space, according to the importance that each requirement has with respect to the app context (e.g. importance of the presence of a given device).

This way, a proximity measure could be created to quantify the distance between the point representing the app and each point representing a technological stack. The technological stack which minimizes the distance from the app is the first candidate to be the best fit to the considered app context. The feasibility of mapping technological stacks and app requirements as points in the previously defined space can follow the Impact Vector Model; in particular, the app requirements can be modeled as a “goal impact vector” (i.e. the expected result of the development), while technological stacks can be modeled as process impact vectors (i.e. the possible combinations of languages, approaches, platforms and tools that can be used for the development). For brevity, we omit the details of the mapping of app and technological stacks as points in the space. However, a proximity measure still has to be defined, so that a Tech Expert can apply our model in a real context; a definition of such proximity measure and an example of use of our model are provided in Section 8.

XX.v.1. Implementation and Graphical View of the Model

There are both numerical and graphical problems to represent a space with more than three dimensions. Concerning the first, discrete dimensions and Ordinal scales rather than Real scales are possible for the interesting functions, with consequent limits on the mathematics that can be applied when looking for optimal solutions under defined constraints. Concerning the last, our decision was to start by using a simple user interface; in fact, the focus of this paper is on (i) exploring the mobile development technology modeling for decision-making; and (ii) empirically verifying models in lab and on the field. This is the reason why we did not put much
effort on building up an advanced user interface for our tool prototype. In particular, the
prototype is based on a table (see Table 1) which implements the space defined in Section 5. In
order to model the five dimensions in a flat surface – which allows an easy use of the model, –
the item type of such table (i.e. a point in the space) represents more than one piece of
information; specifically, each item of the table is itself multi-dimensional, so creating a
topological space similar to a manifold [93]. Each elementary item of the table (i.e. each
dimension of a given point) expresses a measure in a four-value Ordinal scale, which are
graphically represented by the symbols ◦, ■, □, and ◻, where ◻ denotes the null value, i.e.,
“Not supported”.

In practice, the table represents a complex but homogeneous abstract entity, that is a 5-cube,
whose elements are symbols of the given ordinal scale.

Table 1 is structured in two quadrants. These realize two guides for selecting the technology to
use according to: a) the needs, and b) the device-specific hardware features that the particular
technology offers, respectively.

The first quadrant implements the subspace (DT, NFD, DA, PC); the abscissa reports the
development technologies, the ordinate represents the needs. The type of development
approach, as leveraged by the corresponding development technology, is represented by
different columns and different background colors: (i) Web in the first column (Orange); Native
in the subsequent triple of columns (Blue); and Hybrid in the last triple of columns (Green).
Each item shows a value in the given Ordinal scale. Since the values reported do not depend on
the platforms specificities for each given development technology, such platforms are not
represented explicitly. In practice, this quadrant represents a cube with colored slices; each
element of the cube includes as many times the ordinal measure, shown in the first quadrant,
as the number of platforms.

The last quadrant shares the abscissa (DT) with the first quadrant, while the ordinate
represents the device features. Many items in this quadrant represent arrays of platforms
(rather than any platform as in the first quadrant). The reason is that each technology can offer
different features depending on the platform on which it is used, and a measure is expected for
each of those platforms. Again, the actual platforms that are considered separately are iOS and
Android, whereas the remaining platforms are grouped into the category “Others”.

---

The scale elements assume a slightly different meaning in the two quadrants. In the first quadrant, the semantic of the scale is \{Null, Insufficient, Sufficient, Excellent\}; in fact, the symbols indicate the extent to which, for the considered development approach, that specific technology satisfies the corresponding need: \(\bullet\) insufficiently, \(\bullet\) sufficiently, and \(\bullet\) excellently. In the last quadrant, instead, the semantic of the scale is \{Null, Rough, Medium, Well\}; in fact, those symbols indicate whether the specific technology provides APIs for using the corresponding device feature, and their level of support: “Well”, i.e., supported for all the different versions of the same platform, “Medium”, i.e., supported for some but not all the
different versions of the same platform”, or “Rough”, i.e. lightly supported. In the first quadrant we can find only one symbol per table item, whereas in the last quadrant, as already mentioned, an item is an array of symbols (i.e. a 4-dimension point), and can also include all symbols together (one per point dimension).

We filled out the dimensions of the matrix discussed above by collecting information from various scientific papers, personal experiences (first quadrant), and technical online documentation (last quadrant). Table 1 also synthesizes on the results from this work.

**XX.vi. The Process of Technology Decision Making for Mobile**

As already mentioned, we want to develop a guide to select the best technology to use for the development of a specified mobile application in a given context. For this reason we can say that our output is the Technology dimension, while the other dimensions of Table 1 are our inputs.

Probably, a multivariate analysis should be used for identifying the best technology to exploit under the given requirements and constraints. However, at this stage of our work, since there is no way to apply multivariate with the small and artificial dataset available, our decision was to start by using simpler techniques.

In general we have two kinds of requirements: General Requirements (GRs), which are present in any mobile app, and Specific Requirements (SRs), which are explicitly stated for the current app, and are subtypes of the GRs.

In order to enact a technology selection process, the Tech Experts are required to assign a weight to each GR, based on its importance or relevance for any app to develop, and then to give a score to each requirement. Concerning the latter, let N be the number of points to distribute across needs and device features. In the setting that we used for our approach, 25% of such points are automatically and evenly assigned to the GRs, and the remaining 75% of the N points are left for assignment to Tech Leaders; these will assign them as they prefer to the SRs, based on their assessment of the requirement relevance for the app being developed. The initial 25% assignment is made to diversify the obtained results, without losing the contributions that other features give to the decision making. This point distribution can be changed arbitrarily, without affecting the model validity.

Finally, Tech Leaders enter Table 1 and obtain, as a result, a value in the previously defined Ordinal scale for each technology. In order get more manageable results, they can make the further decisions of translating those values in the notation of Real numbers, e.g., by using a locally defined translation map; this allows to switch from an Ordinal scale to a Real measurement model, which enables to apply the algebra of Real numbers. At this point, each
generated number represents a numerical score which should quantify numerically the capability of each technology to fulfill any given requirement.

We are aware that the mentioned scale transformation is a theoretical and practical hazard [92, 94]. However, also in this case, the Impact Vector Model supports us to combine and manage heterogeneous dimensions.

**XX.vii. Case study**

As an example of using our matrix for decision making, let us consider a synthetic version of a case study we conducted. Let us suppose that a Tech Leader is requested to develop an application with the requirements shown in the Table 5, row 1.

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The app is required to allow an user to chat with other people connected to network by the same application. Moreover, the app is required to provide the user with the ability of filing and sending recorded audio clips and video messages. Furthermore, the app is required to allow an user to extend her/his contact list by specifying the telephone number of the person s/he wants to include; her/his mobile phone list of contacts is expected to be allowed as a source for such a phone number.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Available on Android and iOS</th>
<th>Notification Alert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to media</td>
<td>Access to videocamera</td>
<td>High performance</td>
</tr>
<tr>
<td>Access to local files</td>
<td>Access to contacts list</td>
<td>Deployable on app stores</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cheap development costs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Access to network</td>
</tr>
</tbody>
</table>

Ten features are explicitly stated as significant for this application, as shown and numbered from 1 to 10. Let the Tech Leader assign the same weight to all general requirements (GRs), and assuming N=1000, to distribute the remaining 750 points on each specific feature (SRs) in the following way:

1. 100 points, 2. 30 points, 3. 10 points, 4. 60 points, 5. 150 points,
6. 30 points, 7. 80 points, 8. 70 points, 9. 70 points, 10. 150 points.
Additionally, in order to have manageable results, let the Tech Leader map the given Ordinal scale into a Real scale, as explained in the previous Section, by using the following map:

- Excellent: 2.0 points.
- Sufficient: 1.0 point.
- Insufficient: 0.5 points.
- Not Supported: 1.0 point.

The Tech Leader can now enter the matrix, and obtain an ordinal score for each technology. Subsequently, he can enter the map, and translate the ordinal symbols found into Real numbers; eventually the results shown in Figure 57 will be obtained.

As we can see, the two technologies with the highest scores are:

1. Platform Specific Development Kit with 1675.9 points.
2. Adobe Flex Mobile + Air with 1491.05 points.

For the given context, the more meaningful definition of proximity measure is the vector distance from the optimal vector (i.e. the resulting difference vector). The optimal point is the one with maximum achievable score, which is 2000 points (i.e. excellent in every dimension), which exactly matches the expected characteristic of our app. In practice, for our context, minimizing such distance is equivalent to pick the maximum among the computed technology scores. Based on the scores shown in the Table 3, the best technological choice for this
application should be the native approach with the Platform Specific SDK. A good choice would be also the Adobe Flex Mobile + Air, in fact, if there are heavy constraints on the time to market and there is not enough in-house skills for each platform (which is a characteristic not represented in our model), such technology could be the only feasible way.

In both cases, however, developers are given all the information required to make an informed choice on what technology best fits the application to develop in the given context.

XX.viii. Validity Issues, Solution Limitations, and Lesson Learned

The model proposed in the previous paragraphs was populated by interviews with experts of, and tested in four case studies within, the IBM Italy Rome Smart Solutions Lab. Such case studies were concerned with different problems to address and they covered a wide range of possibilities, but in a limited number of domains (e.g., mobile banking, education, and mobile marketing).

Once minded that we are coping with an exploratory study, based on feedback that we had from practitioners, the solution validity of the study, i.e., its internal validity, can be considered high enough; on one side, this is concerned with the solution usefulness (does it address relevant practical problems?), which was very positively evaluated by the involved experts; on the other hand, it is concerned with the goodness (how well the solution is constructed); in the opinion of the same practitioners, practical and actionable results were produced by the study, even in absence of a strong and theoretically rigorous context.

In our view, the proposed solution is portable to, and promises to be effective in, different contexts and cases. In other words, the external validity of the proposed solution is considerably high. However, due to the exploratory nature of the study, consequential limitations should be taken into consideration.

First of all, it is important to remark that interviews, as source of collected data and knowledge, involved quite a limited number of participants in the role of mobile technology experts. Also, only one subject developed the four case studies that we have been performing. Additionally, the proposed model was piloted with a single organization. This increased the threat of having the same person repeating all positive or negative actions from case to case, and people doing the technology evaluation in the same context.

One more aspect, which was not considered in this study, and which affects the external validity and limits of the proposed solution, relates to the communication model utilized by the app to develop, e.g., apps connecting more than one partners synchronously and/or asynchronously. This includes considering if, and to which extent, media-content/social networks would influence
the development of mobile apps, and how this should be modeled by an enhanced version of the proposed solution.

Again concerning threats on validity and limits of the proposed solution, we notice that our model is oriented to supporting project leaders in decision making, but it is based only on technical attributes. Constraints placed by the strategic management should be introduced in the model, to support technicians to undertake value-based decisions [95], including risk and return on investment, and proceed coherently with the strategic goals of their company.

A further relevant aspect is that the information gathered during interviews was mainly qualitative knowledge; additionally, measures in Ordinal scale, as produced from case studies, were eventually influenced by the subjectivity of the involved experts.

In our understanding, in case of exploratory studies, which is our case, it is reasonable to accept all the aforementioned validity threats, given the mitigation strategies we enacted, as usual in the experimentation in the field of Software Engineering [80].

For what concerns the lesson learned, let us recall that interviews with experts helped populate the matrix. However, as the case studies proceeded, we had to include additional attributes in the model (i.e. dimensions in the space) and refine the seeded values. This confirmed our conviction that an iterative-incremental approach should be enacted to define the model and populate its base of knowledge. Following the initial training of the data and knowledge base, as enacted by technology experts, the enrichment of the base should be enacted continually, based on decisions made by the organization’s Tech Leaders, possibly leveraging a structured methodology, e.g. the Quality Improvement Paradigm [96]. Additionally, each development organization might need to manage its own base of knowledge for mobile apps. These solutions should also help solving expected dependencies of the model from the context of the organization, e.g., by realizing organization-wide dynamic bases of knowledge refined by domain (mobile health, mobile education, etc.).

Chapter XXI. Exploiting Software Inspection Data for Early Defect Removal

On the path of evaluating the progress Impact One is making with process improvement via impact vectors, in this chapter we want investigate how inspections are performing with respect to defect removal capability and we suppose that several years are gone since inspections were adopted.
A long history of experience and experimentation has produced a significant body of knowledge concerning the proven effectiveness of software inspections. Data and experience from many years and many types of organizations have shown that a properly conducted inspection can remove between 60% and 90% of the existing defects [97]. It is well established that inspections in the earliest phases of software development yield the most savings by avoiding downstream rework.

However, the value of inspections varies widely both within and across organizations. Inspection effectiveness and efficiency can be measured in numerous ways (defects found, defects slipped to testing, time spent, defects found per unit of effort, etc.), and may be affected by a variety of factors, some related to inspection planning (e.g. number of inspectors) and others related to the software (e.g. languages used for the artifacts) and the type and structure of the development organization. The work described here is based on an analysis of a large body of data collected from inspections at Impact-One.

Software engineers at Impact-One quickly realized that inspection planners would benefit from guidance on how to manipulate the variables they could control to maximize the performance of the inspection. The result was the definition of a set of heuristics based on data from a large number of inspections they conducted. These heuristics have been widely disseminated across all the Impact-One teams and provide guidelines for the following factors:

- **Team size**, the number of participants involved in the inspection; small teams are likely to lack important perspectives, while larger teams are more likely to experience dynamics that limit full participation.

- **Meeting length**; if meetings stretch on too long, members’ energy is likely to flag and the results are likely to be less than optimal. When an inspection requires a big amount of work, it is recommended that additional meetings be scheduled.

- **Page rate**, the number of document pages that the inspectors examine per hour of the meeting; giving a team too much material to look through will invariably result in a more superficial inspection.

Heuristics may vary based on the type of artifact being inspected, which represents a factor of mainly importance during inspection planning. Based on the repository available to us, we identified four artifact types: requirements, design, code, and test. For each artifact type, an analysis and synthesis of existing heuristics and studies at Impact-One led to the values reported in Table 6.

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Team size</th>
<th>Meeting length</th>
<th>Page rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>[5; 8]</td>
<td>~2</td>
<td>[10; 30]</td>
</tr>
</tbody>
</table>

In the formulation and use of these heuristics, the outcome variable of interest has been the total number of defects found during an inspection. There have been both historical and practical reasons for this focus. During inspection planning, the total number of defects in the software to be inspected is unknown. It is also, practically speaking, unknowable, in this environment, due both to the variety of verification and validation activities applied to the software, and the difficulty in reconciling data from these activities with inspection data. Thus, managers and inspection planners have focused on maximizing the number of defects found in an inspection, and using that as a benchmark for comparing the effectiveness of different values for the inspection planning parameters (i.e. team size, meeting length, and page rate).

In this chapter, we investigate whether, and to which extent, complying with the given heuristic actually turns into benefits in terms of the stated goal of maximizing the number of detected defects, i.e., the efficacy of the inspection.

As we will be showing, compliance with the heuristics has a significant positive impact on the performance, i.e., the effectiveness, of the inspection; therefore, management should reaffirm and encourage inspection planners to use the provided guidelines. On the other side, as a result of our study, we found that, in some conditions, compliance with those heuristics is not necessarily the best decision driver, because more compliance sometimes may cause higher cost with no practical benefits.

In the rest of this chapter, we describe the state of the practice of inspections at Impact-One and make explicit the research question we address. Then we present the results of our analysis, along with the methodology we created and the decision model we built in order to package our current knowledge on inspections at Impact-One. Afterward, we will discuss the threats of our empirical investigation and eventually draw some conclusions.

**XXI.i. Inspections at Impact-One: State of the Practice**

In order to investigate the performance of existing heuristics for inspection planning, we accessed a large repository containing more than 2500 records, each representing an inspection in terms of artifact, team size (TS), meeting length (ML), number of detected defects (DD) and more attributes. Data were collected across all teams centers over several years, but it
is important to remark that more recent inspections behave equivalently to older inspections in terms of validity of the heuristics, which is the focus of our analysis.

Completeness of records represented an issue we had to cope with, as some fields were sometimes left blank. We created a list of exclusion criteria to apply to the repository in order to obtain usable data. In particular, a record should be filtered out when:

- the number of detected defects is unknown (it does not provide useful information)
- the number of participants is:
  - unknown (level of compliance, LC, unknown)
  - 1 (no inspection meeting, therefore no structured inspection)
- the meeting length is
  - 0 (no inspection meeting, therefore no structured inspection)
  - unknown (LC unknown)
- page rate is unknown (LC unknown)
- product type is “other” (heuristics are provided on a by-artifact basis)

After filtering, the number of remaining records was 611, structured as follow: 15 on requirements, 190 on design, 302 on code, and 36 on test artifacts. Additional statistically relevant information is reported in Table 7.

### Table 7 Descriptive statistics

<table>
<thead>
<tr>
<th></th>
<th>Team size</th>
<th>Meeting length</th>
<th>Page rate</th>
<th>Detected defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>3.724</td>
<td>0.885</td>
<td>59.241</td>
<td>8.403</td>
</tr>
<tr>
<td>Median</td>
<td>3.000</td>
<td>0.500</td>
<td>32.867</td>
<td>4.000</td>
</tr>
<tr>
<td>Mode</td>
<td>3.000</td>
<td>0.250</td>
<td>24.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>1.485</td>
<td>0.750</td>
<td>127.621</td>
<td>10.046</td>
</tr>
<tr>
<td>Minimum</td>
<td>2.000</td>
<td>0.100</td>
<td>0.267</td>
<td>0.000</td>
</tr>
<tr>
<td>Maximum</td>
<td>8.000</td>
<td>4.000</td>
<td>2423.600</td>
<td>59.000</td>
</tr>
</tbody>
</table>

The analysis of such a large repository revealed that the higher the compliance with heuristics (see Table 6), the higher the number of defects detected during the inspection (see Table 8, columns 1 and 2). From further analysis, it is also noticeable that only ~10% of teams apply the inspections at the highest compliance with the given heuristics (Table 8, columns 3) and that almost two thirds of the inspections have been performed complying with one or zero heuristics, which correlates to a lower number of detected defects.
Table 8 Behavior of teams with respect to heuristics

<table>
<thead>
<tr>
<th>Number of heuristics the inspection complies with</th>
<th>Average number of detected defects</th>
<th>Sample size</th>
<th>Ratio of inspections</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>19.965</td>
<td>64</td>
<td>10.474%</td>
</tr>
<tr>
<td>2</td>
<td>12.481</td>
<td>145</td>
<td>23.732%</td>
</tr>
<tr>
<td>1</td>
<td>6.076</td>
<td>266</td>
<td>43.535%</td>
</tr>
<tr>
<td>0</td>
<td>3.149</td>
<td>136</td>
<td>22.259%</td>
</tr>
</tbody>
</table>

Figure 58 offers a graphical representation of the same information, where the size of the spheres represents the number of inspections performed at the corresponding level of compliance (i.e. the sample size).

These qualitative considerations are supported by our statistical analysis, which we now synthesize.

Let us primarily define the metric “Level of Compliance”.

Def: Given an inspection on a particular artifact, its Level of Compliance, LC, is the number of inspection attributes (i.e. team size, meeting length and page rate) compliant with the heuristics for the corresponding artifact type. The level of compliance for an inspection is $n$ when exactly $n$ attributes are compliant with the heuristics; therefore, $n \in \{0; 1; 2; 3\}$. If some parameters are unknown, then the level of compliance is undefined.
In the dataset we used for the analysis, we utilized the number of detected defects which records are complete, i.e., no parameters is unknown.

The first step of the statistical analysis is to run the Jonckheere–Terpstra test (or Jonckheere’s trend test, non-parametric): given an ordered list of levels for the factor (LC in our case), the Jonckheere–Terpstra test can detect the existence of a monotonic increasing trend in the response variable when the factor increases. More specifically, the test rejects the null hypothesis if at least one pair in this list of levels shows a statistically significant difference on medians with the next level. In our case, the null hypothesis of the Jonckheere–Terpstra test is rejected with p-value smaller than 0.001; this means that increasing LC leads to increasing the number of detected defects.

Subsequently, we investigate each pair of consecutive LCs, in order to identify which pairs are significantly different. Results of this set of 3 comparisons are reported in Table 9: samples are all normally distributed and independent, thus we ran 4 unpaired t-tests.

<table>
<thead>
<tr>
<th>#</th>
<th>Comparison</th>
<th>p-value</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LC=0 vs LC=1</td>
<td>&lt; 0.001</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>LC=1 vs LC=2</td>
<td>&lt; 0.001</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>LC=2 vs LC=3</td>
<td>0.001</td>
<td>Y</td>
</tr>
</tbody>
</table>

All tests detect the existence of significant differences between pairs. Therefore, it is evident that teams should be encouraged to increase their compliance with heuristics, if they aim to detect the highest number of defects. On the other hand, complying with the heuristics often represents a cost (e.g. increasing the number of inspections or stretching the meeting length). Also, in some cases, constraints may exist limiting the possibility of complying with all heuristics.

How can inspection planners identify the tradeoff between cost and compliance? Since compliance has been shown to correlate to performance, this project management question assumes a particular relevance and, in this paper, we try to address it by answering the following research question: are the heuristics accurate enough to provide the adequate level of support to inspection planners?

As a result of our preliminary examination on the state of the practice, heuristics represent a good guideline, but teams seem not to be enabled to exploit their entire potential; in fact, most of the times they do not (or cannot) behave according to them. Consequently, in the rest of this paper, we will try to build a new and finer-grained decision support for inspection planners that accounts for both inspection performance and cost. This way, we expect to provide inspection
planners with heuristics at the adequate accuracy to maximize the number of detected defects in respect of the context constraints.

XXI.i.1. Decision Model: Statistical Analysis

XXI.i.1. Metrics

In addition to the previously defined “Level of Compliance” (LC), we define the metric “Type of Compliance” (TC).

Def: Given an inspection, its Type of Compliance, TC, is the set of inspection attributes compliant with the heuristics.

Notice that, for the same LC, multiple types of compliance can exist. In particular: for LC=1, an inspection can comply with the heuristic on TS, ML or PR (mutually exclusive); for LC=2, it can comply with the heuristics on TS and ML, TS and PR, or ML and PR.

XXI.i.2. Statistical Tests

In order to define a finer-grained decision model, which is the goal of our analysis, we want to investigate the differences between the various types of compliance across the levels of compliance.

As a first step, we compare consecutive levels of compliance for all TCs; p-values of the statistical tests are reported in Table 10 (significance level is 97.5%). Depending on the normality or non-normality of samples, tests used are, respectively, unpaired t-test (U) or Mann-Whitney test (M).

Table 10 Comparisons between consecutive LCs

<table>
<thead>
<tr>
<th>#</th>
<th>TC comparison</th>
<th>p-value</th>
<th>Significance</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>{} vs {TS}</td>
<td>0.001</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>5</td>
<td>{} vs {ML}</td>
<td>&lt;0.001</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>6</td>
<td>{} vs {PR}</td>
<td>0.003</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>7</td>
<td>{TS} vs {TS, ML}</td>
<td>&lt;0.001</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>8</td>
<td>{TS} vs {TS, PR}</td>
<td>0.732</td>
<td>N</td>
<td>U</td>
</tr>
<tr>
<td>9</td>
<td>{ML} vs {TS, ML}</td>
<td>0.243</td>
<td>N</td>
<td>M</td>
</tr>
<tr>
<td>10</td>
<td>{ML} vs {ML, PR}</td>
<td>0.883</td>
<td>N</td>
<td>U</td>
</tr>
<tr>
<td>11</td>
<td>{PR} vs {TS, PR}</td>
<td>0.393</td>
<td>N</td>
<td>M</td>
</tr>
<tr>
<td>12</td>
<td>{PR} vs {ML, PR}</td>
<td>0.203</td>
<td>N</td>
<td>U</td>
</tr>
<tr>
<td>13</td>
<td>{TS, ML} vs {TS, ML, PR}</td>
<td>0.013</td>
<td>Y</td>
<td>M</td>
</tr>
</tbody>
</table>
As a second step, we analyze the comparisons between non-consecutive levels of compliance, i.e.: LC=0 vs LC=2 (all 3 TCs), LC=0 vs LC=3, and LC=1 (all TCs) vs LC=3. Results are reported in Table 11. As well as for the first set of tests, one-tailed Mann-Whitney or unpaired t-test is used, depending on the normality of samples.

### Table 11 Comparisons between non-consecutive LCs

<table>
<thead>
<tr>
<th>#</th>
<th>TC comparison</th>
<th>p-value</th>
<th>Significance</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>{} vs {TS, ML}</td>
<td>&lt;0.001</td>
<td>N</td>
<td>U</td>
</tr>
<tr>
<td>17</td>
<td>{} vs {TS, PR}</td>
<td>0.186</td>
<td>N</td>
<td>U</td>
</tr>
<tr>
<td>18</td>
<td>{} vs {ML, PR}</td>
<td>&lt;0.001</td>
<td>Y</td>
<td>M</td>
</tr>
<tr>
<td>19</td>
<td>{} vs {TS, ML, PR}</td>
<td>&lt;0.001</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>20</td>
<td>{TS} vs {TS, ML, PR}</td>
<td>&lt;0.001</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>21</td>
<td>{ML} vs {TS, ML, PR}</td>
<td>0.005</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>{PR} vs {TS, ML, PR}</td>
<td>&lt;0.001</td>
<td>Y</td>
<td>U</td>
</tr>
</tbody>
</table>

It is interesting to notice that increasing the level of compliance from 0 to 1 always leads to a performance improvement (comparisons #4, #5 and #6), as well as increasing it from 2 to 3 (comparisons #13, #14 and #15). However, this is not true when increasing LC from 1 to 2, because a statistically significant difference exists only for the comparison TC=\{TS\} vs. \{TS, ML\}.

It is also interesting to notice some heterogeneity in results: in fact, increasing LC does not systematically lead to an increase in effectiveness, because the existence of such an increase depends on the specific types of compliance. In other words, even though results show that, in the average, the more compliance leads to higher effectiveness, a more detailed analysis demonstrates that the level of compliance per se is not the best decision driver one can adopt in the context under study; rather, a case-by-case decision could be more appropriate.

As a third step, we analyze the comparisons between different TC within the same LC.

### Table 12 Comparisons within the same LC

<table>
<thead>
<tr>
<th>#</th>
<th>TC comparison</th>
<th>p-value</th>
<th>Significance</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>{TS} vs {ML}</td>
<td>&lt;0.001</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>{TS} vs {PR}</td>
<td>0.718</td>
<td>N</td>
<td>U</td>
</tr>
</tbody>
</table>
The third and last set of comparisons reveals that, within the same LC, it does make a difference the TC selected, i.e. different TCs lead to different expected performance.

**XXI.iii. Decision Model: Interpretation of Results**

In order to provide the interpretation of results in Table 10 and Table 11, we propose the decision model depicted in Figure 59; a state represents a TC and an edge represents a statistically significant difference between source and destination. The direction of the edge indicates the direction of the difference, with the arrow pointing at the higher performance.

![Figure 59 State-based decision model](image-url)
An inspection planner can easily leverage the proposed model during the inspection planning. In fact, the usual decision methodology can be applied to set team size, meeting length and page rate of the inspection to undertake. Afterward, based on the proposed attribute values for the attributes and according to the heuristics, the initial state on the decision model can be identified. For example, given an inspection on a requirements document of 30 pages, for which the inspection planner proposes to allocate 3 people and 2 hours, the initial state, according to Table 6, would be TC={ML, PR} (see Figure 60 and Figure 61).

Once the initial state has been identified, results of our analysis can be immediately exploited by exploring all outgoing edges of the initial state: if there are no outgoing edges, then the current inspection attribute configuration maximizes the expected performance. If there are some outgoing edges, then any of them can be selected (e.g. see Figure 61); selecting an edge corresponds, in practice, to modifying the inspection configuration by changing one or more attribute values that are not compliant with the heuristics. By changing such attribute values to be compliant with the corresponding heuristic takes the inspection configuration to a new state that, in the average, corresponds to significantly better performance. From this new reached state, the same reasoning can be applied to consider further configuration improvement, whereas the decision maker is willing to invest more resources in order to increase the likelihood of getting higher performance.

Taking on the previous example, there is an outgoing edge the inspection planner can consider; such edge goes from TC= {ML, PR} to TC={TS, ML, PR}: in practical terms, complying with the heuristic on TS by including at least two more inspectors (from 4 to 6) leads to higher expected number of detected defects for the given inspection.
Our model can be useful also when constraints exist on the resources available to the inspection planner. For example, given a constraint on the values of the attributes, if the admissible values are all not compliant with the heuristics given for the artifact selected for inspection, then all states including the compliance with that attribute should not be reached; in other words, the original decision model can be immediately tailored to the context and reduced to having fewer states and edges. Complementarily, in case the constraint forces the attribute value to be compliant with the heuristic, all states not requiring such compliance should be excluded by the tailored model.

For example, if the number of available inspectors for the previous example is 5, then the state-based model to be used is the one in Figure 62. In this case, the configuration proposed by the inspection planner is the one that maximizes the expected number of detected defects.

**XXI.iv. Threats**

According to the classification of validity threats in [98], a conclusion validity threat we need to account for is the random heterogeneity of subjects. In fact, inspections were performed in different centers, thus different people with potentially different skills and capabilities may have produced the inspection data we investigated. However, based on the results reported in [97] regarding an analysis of the same dataset, we have found no systematic bias according to the center.

Internal validity of the experimental analysis is threatened by the fact that many variables can influence the inspection performance, but we investigated only some of them. As already mentioned in section 1, many previous investigations identified team size, meeting length, page rate and artifact type as influential factors of an inspection, i.e. variables that strongly correlate to the inspection performance. Additional variables can also be relevant, e.g.: language of the artifact, expertise of the inspectors, quality of the artifact, domain of the system under analysis and so on. When the number of variables increases, the complexity of the study becomes problematic: even in presence of large datasets, as the one available to us and used for the present study, an experimental investigation accounting for all potentially influential variables would become unfeasible and statistically (and practically) unreliable. Therefore, the way out we opted for has been picking the most influential variables and focus the investigation on them.
Concerning the external validity, it is obvious that narrowing the domain limits the exportability of the results to other contexts. However, it is remarkable that, even though results are not applicable to contexts different from Impact-One, the applied methodology is totally reusable: replacing the dataset enables teams in diverse contexts to build a state-based model similarly to the one represented in Figure 59 to support inspection planning. In particular, such a new model will be based on different heuristics (according to context-specific guidelines and procedures) and will have different edges as a result of the statistical analysis; nevertheless, its creation and use will not change.

Chapter XXII. Overall Experimentation Threats

In the previous chapters, we reported on several empirical investigations we performed to evaluate the impact vector model. As me stated, our abstract and synthetic company is a way of putting together our investigations and creating a leit motiv for showing the potential of our model. Nevertheless, even though we reported validity threats on a per-experimentation basis, there are some more threats we need to report concerning our general approach to validation.

First of all, we did not actually have the chance to validate the impact vector model in its entirety in a single real company. This may trigger some doubts on the real applicability of the model in real contexts. However, as we’ve shown, all parts of the model seem sustainable and feasible in practice, when used in isolation; furthermore we made plans for the extensive application of our model for the next few years in some of those contexts and most of them were positively accepted by our partners. Therefore, we plan to be able to report on them to further mitigate the doubts on an encompassing and pervasive application of the impact vector model.

A second threat applies in general to our experimentation: each part of the model has been validated in at most two different situations. Actually, only leveraging the impact vectors was validated twice (one for testing improvement and one for inspection tuning), while the remaining parts of our framework were the objects of single investigations. Obviously, this represents a limitation of our experimentation. However, such types of empirical studies are expensive and risky for the organizations running them; therefore, they require a lot of preparation and significant investments. We struggled and worked hard to be able to experiment every part of the model, but replication has rarely been an option for us.

A third threat, which correlates to the second, concerns the domains of application: different parts of our model were validated in different domains. Even though, on one side, this represent a pro – as it is an indicator of the flexibility of the model – on the other side it
represents a threat, because we cannot be sure that every part of the model is feasible and reasonable in contexts other than the one where that part was validated. However, during the empirical investigation reports, we did not need to precisely describe every single real domain and context, but we were able to contextualize the experimentation within our abstract company. This is a good indicator of the fact that the model, though flexible and tailorable, does not need to be modified in its foundation to accommodate context-specific needs; consequently, the same approach, the same process, the same tools and the same concepts can be totally reused across different domains and contexts.

Summarizing, we recognize that the validation of our impact vector model is still at its beginning, but we can also affirm that we earned the confidence of some of our partners and we will be able to further investigate the potential of our approach on the field.
Section VI: Final Thoughts and Future Work
Chapter XXIII. **Final Thoughts**

In this work, we have defined, explored and investigated an approach to managing software development. This approach is named Impact Vector Model and it is part of the Impact Vector Framework, whose definition is also included in this Ph.D. thesis. Our approach is evidence-based, value-based and goal-driven. Our target audience consists in decision makers at all levels, as it enables dealing with both business and technical aspects of software development, and it actually tries to integrate these two perspectives for making decision both technically and economically sound and convenient. The goal of our model is to support decision makers during their management activities, e.g. planning, monitoring, measuring, predicting, correcting expectations, acquiring or freeing resources, prioritizing, selecting, evaluating, assessing and so on. Other than practically usable, we outlined a theoretical foundation for our model, necessary to give a rigorous and clear structure to experimental data that the model requires to deal with. In conformance with an engineered, scientific, mid-term work, we went through many iterations of definition-experimentation-refinement, with the collaboration of industry and academia to come up with well-founded and concretely useful artifacts and knowledge. Results encourage us to proceed on our path and to expand the boundaries of our work to including more domains, more processes, more partners, more data and more models. In the last few pages of this work, we will sketch out some future and current work that has to be performed in order to transform the Impact Vector Model into a reliable approach to software development measurement and management.

Chapter XXIV. **Future Work**

**XXIV.i.1. Consolidating the Impact Vector Framework**

As shown in Section IV, we were able to propose a tool chain to support the implementation of the Impact Vectors in practice. However, the tools of the chain are somehow disconnected and only a wise user willing to leverage the impact vector model can actually find the path through so many different tools. Therefore, one of the first goals we plan for the next future is to consolidate the tool chain by providing a set of bridges and connectors to facilitate the collaboration among those tools. Concretely, this turns into some technical refinements, e.g. the migration of IV-Schema from MySql to DB2 in order to directly support its integration with Cognos Insight, but it also requires some more substantial intervention. Probably the hardest and expensive task will be the extension of Rational Method Composer to support the use of Impact Vectors in the IBM vision of software development process management. Since that tool is commercial and continuously evolving, the realization of a similar extension very likely
requires the contribution of the vendor itself; with this aim, we have already initiated to explore the feasibility of this task and received some positive feedbacks.

**XXIV.i.2. Impact Vector Model and Empirical Software Engineering 2.0**

For many years empirical software engineering, by leveraging a number of different analysis and synthesis techniques, has been supporting practitioners and researchers to assess, verify, qualify, and improve software processes, products and technologies. Aggregating data from many experiments, performed in different contexts, with different methods and objects, and by different people, has always been a challenge. Examples are CeBase (no longer ongoing) and Promise, repositories of Empirical Software Engineering data; the goal of such repositories was to enable global sharing of data to explore and analyze, in order to monitor, plan and predict SE practices. Even though the importance of empirical data is widely recognized, many times they are not shared and sometimes not even exploited; there might be many reasons to why such global approaches may not work, e.g. the sensitivity of industrial data or the lack of incentives to stimulate data sharing. However, maybe pushed by some romantic vision, we believe that sharing data is a moral duty of companies operating world-wide and advancing the state of the practice with their everyday work; also, researchers, with the same spirit, should be natively encouraged to share their results and provide more and deeper insights in the research branches they investigate.

Based on this, and on the path lightened by Tim Menzies of Empirical Software Engineering 2.0, our contribution is to provide a flexible, extensible, tailorable, formally well-founded structure of the data to be shared among researchers and practitioners. As depicted in the Section IV, we are building a platform to encourage and guide researchers and practitioners to contribute to advance the Empirical Software Engineering by using the Impact Vector Model to organize and share data in a reusable and valuable way. In the short future, we plan to deploy IV-Schema, the support tool we are building, and enable people to contribute to advancing the state of Software Engineering.

**XXIV.i.3. Business Analytics and the Impact Vector Framework**

While gaining confidence with IBM Cognos Insight and its capabilities, we found that in the area of Business Analytics, and, more specifically, in the branch of Business Intelligence, many people are elaborating and working on a huge amount of economic data. Their tasks often require tuning and optimization of business processes as the result of their quantitative and qualitative analyses of the available data. Often, their goal ends up being the maximization of the profit, but inherently it consists of a number of sub-goals that could be brought back to something similar to the ones faced by Software Engineers during process planning and software development process performance enhancement. We believe that a careful
investigation of similarities and differences between the two areas (business analytics and software development process enhancement) might give birth to interesting synergies, as well as exchange of opportunities, ideas and methods in both directions. What we also consider an indicator of this potential is the field of Business Process Modeling (BPM), which is currently being paid attention by the OMG (as an example, see standard on BPMN). BPM aims at modeling business processes not strictly related to software. However, looking at BPM representations (e.g. BPMN) and the process it is applied through, it is evident to any decent Software Engineer that there are similarities, other than an obvious difference in terms of focus between BPM and software development process (trivially, the one on business management, the latter on software development). Just to mention one, in both cases some requirements (business or software) are identified and defined, then graphically represented (e.g. BPMN or UML), and then transformed in something executable; one more interesting aspect is the existence of UML profiles for BPMN, as well as the interest of the same computer-related entity, i.e. the OMG, on both themes. Such clues are valuable to us and encourage us to explore also the BPM field, in order to detect potential synergies that may lead to mutual benefits.

**XXIV.i.4. Further Contexts of Application of the Impact Vector Framework**

In the present work, we mostly reported on verification tasks. This was not entirely a choice, but we were driven into that by our industrial partners, their needs and our current personal fields of interest. Nevertheless, we also sketched out something concerning tasks other than validation-related (see Section IV). In the next future, we plan to expand the field of application and, in particular, of experimentation of Impact Vectors to including many more activities. Always within the group of software verification and validation activities, assurance cases and the broad sub-branch of more formal methods represent an good challenge and an interesting path to explore. Outside verification and validation, instead, risk management and requirements management activities are our next targets: which attributes should be used to accurately characterize the sub-process of requirements management? Which ones for risk management and mitigation?

Also, do we need to refine the Impact Vector Model to accommodate the needs of peculiar processes, e.g. the development of unique pieces of software, like flight software? How can we leverage the Impact Vector Model, if any feasible, to support the work of teams with specific tasks or using particular processes, e.g. Quality Assurance people or Agile teams?

Those are open questions that will probably and hopefully give us much to work for the next few years.
References

References in Alphabetical Order


[76] Eclipse Helios: http://www.eclipse.org/helios/


[67] Li S., Waters R. Nucleotide level detection of cyclobutane pyrimidine dimers using oligonucleotides and magnetic beads to facilitate labelling of DNA fragments incised, at the dimers and chemical sequencing reference ladders. Carcinogenesis. 17, 8, 1549-1552, 1996.


References in Appearance Order


[67] Li S., Waters R. Nucleotide level detection of cyclobutane pyrimidine dimers using oligonucleotides and magnetic beads to facilitate labelling of DNA fragments incised, at the dimers and chemical sequencing reference ladders. Carcinogenesis. 17, 8, 1549-1552, 1996.


[76] Eclipse Helios: http://www.eclipse.org/helios/


List of Papers Investigated during Testing Literature Survey (only titles)

Notice that the following list includes only papers already investigated. A few hundred papers are still being studied and are not included in this list.

- A clustering approach to improving test case prioritization: An industrial case study
- A combinatorial testing strategy for concurrent programs
- A Comparative Study of Software Model Checkers as Unit Testing Tools: An Industrial Case Study
- A Comparison of Tabular Expression-Based Testing Strategies
- A concept analysis inspired greedy algorithm for test suite minimization
- A Controlled Experiment Assessing Test Case Prioritization Techniques via Mutation Faults
- A history-based cost-cognizant test case prioritization technique in regression testing
- A lightweight process for change identification and regression test selection in using COTS components
- A Metaheuristic Approach to Test Sequence Generation for Applications with a GUI
- A novel approach to regression test selection for J2EE applications
- A novel co-evolutionary approach to automatic software bug fixing
- A platform for search-based testing of concurrent software
- A prioritization approach for software test cases based on Bayesian networks
• A relation-based method combining functional and structural testing for test case generation
• A Safe Regression Test Selection Technique for Database-Driven Applications
• A stateful approach to testing monitors in multithreaded programs
• A Theoretical and Empirical Analysis of the Role of Test Sequence Length in Software Testing for Structural Coverage
• A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search
• A uniform random test data generator for path testing
• Adaptive Random Test Case Prioritization
• Adaptive random testing based on distribution metrics
• An approach for qos-aware service composition based on genetic algorithms
• An Approach to Test Data Generation for Killing Multiple Mutants
• An automated approach to reducing test suites for testing retargeted C compilers for embedded systems
• An Empirical Comparison of Automated Generation and Classification Techniques for Object-Oriented Unit Testing
• An Empirical Comparison of Test Suite Reduction Techniques for User-Session-Based Testing of Web Applications
• An empirical study of incorporating cost into test suite reduction and prioritization
• An empirical study of regression test application frequency
• An Empirical Study of Regression Testing Techniques Incorporating Context and Lifecycle Factors and Improved Cost-Benefits Models
• An Empirical Study of Test Case Filtering Techniques Based on Exercising Information Flows
• An Empirical Study of the Effect of Time Constraints on the Cost-Benefits of Regression Testing
• An empirical, path-oriented approach to software analysis and testing
• An Enhanced Test Case Selection Approach for Model-Based Testing: An Industrial Case Study
• An experimental evaluation of weak-branch criterion for class testing
• An experimental study of adaptive testing for software reliability assessment
• An improved meta-heuristic search for constrained interaction testing
• Application of system models in regression test suite prioritization
• Applying Interface-Contract Mutation in Regression Testing of Component-Based Software
• Assessing and Improving State-Based Class Testing: A Series of Experiments
• Assessing, Comparing, and Combining State Machine-Based Testing and Structural Testing: A Series of Experiments
• AUSTIN: A tool for Search Based Software Testing for the C Language and its Evaluation on Deployed Automotive Systems
• Automated Concolic Testing of Smartphone Apps
• Automated generation of test suites from formal specifications of real-time reactive systems
• Automated Security Testing of Web Widget Interactions
• Automatic generation of test cases from Boolean specifications using the MUMCUT strategy
• Automatic Test Data Generation Using Genetic Algorithm and Program Dependence Graphs
• Automatic Test Generation From GUI Applications For Testing Web Services
• Automatic, evolutionary test data generation for dynamic software testing
• Black-box system testing of real-time embedded systems using random and search-based testing
• Call Stack Coverage for Test Suite Reduction
• Call-stack coverage for gui test suite reduction
• Carving and Replaying Differential Unit Test Cases from System Test Cases
• Carving Differential Unit Test Cases from System Test Cases
• Checking Inside the Black Box: Regression Testing Based on Value Spectra Differences
• Checking Inside the Black Box: Regression Testing by Comparing Value Spectra.
• Code coverage-based regression test selection and prioritization in WebKit
• Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization
• Combining Search-based and Adaptive Random Testing Strategies for Environment Model-based Testing of Real-time Embedded Systems
• Configuration selection using code change impact analysis for regression testing
• Constraint based structural testing criteria
• Contract-Based Mutation for Testing Components
• CUTE: a concolic unit testing engine for C
• Data Flow Testing of Service Choreography
• Decreasing the Cost of Mutation Testing with Second Order Mutants
• Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History
• Design and analysis of GUI test-case prioritization using weight-based methods
• Designing and comparing automated test oracles for GUI-based software applications
• Developing a Single Model and Test Prioritization Strategies for Event-Driven Software
• Differential Testing - A New Approach to Change Detection
• Directed Test Generation using Symbolic Grammars
• Directed Test Suite Augmentation: Techniques and Tradeoffs
• Distributing test cases more evenly in adaptive random testing
• Efficient Software Verification: Statistical Testing Using Automated Search
• Efficient unit test case minimization
• Eliminating Harmful Redundancy for Testing-Based Fault Localization Using Test Suite Reduction: An Experimental Study
• Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques
• Empirical Evaluation of the Fault-Detection Effectiveness of Smoke Regression Test Cases for GUI-Based Software
• Empirical studies of a decentralized regression test selection framework for web services
• Empirically Studying the Role of Selection Operators during Search-Based Test Suite Prioritization
• Evaluating improvements to a meta-heuristic search for constrained interaction testing
• Evolutionary generation of test data for many paths coverage based on grouping
• Evolutionary Test Data Generation: A Comparison of Fitness Functions
• Experimental assessment of manual versus tool-based maintenance of GUI-directed test scripts
• Experiments with test case prioritization using relevant slices
• Fault coverage of Constrained Random Test Selection for access control: A formal analysis
• Finding Bugs by Isolating Unit Tests
• Genetic Algorithms for Randomized Unit Testing
• Grouping target paths for evolutionary generation of test data in parallel
• Guided test generation for coverage criteria
• Handling over-fitting in test cost-sensitive decision tree learning by feature selection, smoothing and pruning
• Highly Scalable Multi Objective Test Suite Minimisation Using Graphics Cards
• HOTTest: A model-based test design technique for enhanced testing of domain-specific applications
• How Does Program Structure Impact the Effectiveness of the Crossover Operator in Evolutionary Testing
• Improved Multithreaded Unit Testing
• Improving Effectiveness of Automated Software Testing in the Absence of Specifications
• Improving evolutionary class testing in the presence of non-public method
• Improving Fault Detection Capability by Selectively Retaining Test Cases during Test Suite Reduction
• Improving Structural Testing of Object-Oriented Programs via Integrating Evolutionary Testing and Symbolic Execution
• Improving Test Case Generation for Web Applications Using Automated Interface Discovery
• Incremental Test Generation for Software Product Lines
• Industrial experiences with automated regression testing of a legacy database application
• Industrial Real-Time Regression Testing and Analysis Using Firewalls
• Integrating Techniques and Tools for Testing Automation
• Inter-Context Control-Flow and Data-Flow Test Adequacy Criteria for nesC Applications
• Iterative context bounding for systematic testing of multithreaded programs
• Model-based test prioritization heuristic methods and their evaluation
• Model-Based Testing of Community-Driven Open-Source GUI Applications
• MSeqGen: Object-Oriented Unit-Test Generation via Mining Source Code
• Mutation Operators for Spreadsheets
• Mutation-Driven Generation of Unit Tests and Oracles
• On test suite composition and cost-effective regression testing
• On the Effectiveness of the Test-First Approach to Programming
• On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques
• Ordering Broken Unit Tests for Focused Debugging
• PAT: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs
• Perturbation-based user-input-validation testing of web applications
• Productivity of Test Driven Development: A Controlled Experiment with Professionals
• Prioritizing component compatibility tests via user preferences
• Prioritizing JUnit test cases in absence of coverage information
• Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis
• Prioritizing tests for fault localization through ambiguity group reduction
• Quasi-random testing
• Quota-constrained test-case prioritization for regression testing of service-centric systems
• Random Testing: Theoretical Results and Practical Implications
• Rapid "Crash Testing" for Continuously Evolving GUI-Based Software Applications
• Reachability graph-based test sequence generation for concurrent programs
•Reachability testing of concurrent programs
• Regression testing for component-based software systems by enhancing change information
• Regression testing in Software as a Service: An industrial case study
- Regression Testing UML Designs
- Representation Dependence Testing using Program Inversion
- Results from introducing component-level test automation and Test-Driven Development
- Scalable and Effective Test Generation for Role-Based Access Control Systems
- Scalable Test Data Generation from Multidimensional Models
- Scaling Regression Testing to Large Software Systems
- Search Algorithms for Regression Test Case Prioritization
- Selecting a cost-effective test case prioritization technique
- Should software testers use mutation analysis to augment a test set?
- Size-Constrained Regression Test Case Selection Using Multicriteria Optimization
- Stress Testing RealTime Systems with Genetic Algorithms
- Strong Higher Order Mutation-Based Test Data Generation
- Studying the Fault-Detection Effectiveness of GUI Test Cases for Rapidly Evolving Software
- Test case prioritization using relevant slices
- Test coverage optimization for large code problems
- Test generation via Dynamic Symbolic Execution for mutation testing
- Test Input Generation Using Dynamic Programming
- Test Prioritization Using System Models
- Test suite reduction and prioritization with call trees
- Test Suite Reduction with Selective Redundancy
- Test-case prioritization with model-checkers
- Testing Context-Aware Middleware-Centric Programs: A Data Flow Approach and an RFID-Based Experimentation
- Testing input validation in Web applications through automated model recovery
- The Effects of Time Constraints on Test Case Prioritization: A Series of Controlled Experiments
- The Impact of Input Domain Reduction on Search-Based Test Data Generation
- The State Problem for Test Generation in Simulink
- Towards a distributed execution framework for JUnit test cases
- Towards the prioritization of regression test suites with data flow information
- Traffic-aware stress testing of distributed real-time systems based on UML models using genetic algorithms
- Unit-level test adequacy criteria for visual dataflow languages and a testing methodology
- Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing
- Using hybrid algorithm for Pareto efficient multi-objective test suite minimization
• Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria
• Using random test selection to gain confidence in modified software
• Using the Case-Based Ranking Methodology for Test Case Prioritization
• Utilization of Extended Firewall for Object-Oriented Regression Testing
• XML-manipulating test case prioritization for XML-manipulating services