ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Software design as story telling: Reflecting on the work of Italo Calvino

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Ciancarini P., Masyagin S., Succi G. (2020). Software design as story telling: Reflecting on the work of Italo Calvino. Association for Computing Machinery, Inc [10.1145/3426428.3426925].

Availability:
This version is available at: https://hdl.handle.net/11585/798021 since: 2021-02-10

Published:

DOI: http://doi.org/10.1145/3426428.3426925

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

12 October 2024

This is the final peer-reviewed accepted manuscript of:

Ciancarini, Paolo and Masyagin, Sergey and Succi, Giancarlo, Software Design as Story Telling: Reflecting on the Work of Italo Calvino. Paper presented at the Onward! 2020 - Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, p. 195–208

The final published version is available online at ***https://dx.doi.org/10.1145/3426428.3426925***

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

Software Design as Story Telling: Reflecting on the Work of Italo Calvino

Paolo Ciancarini
Università di Bologna, Italy
Innopolis University, Russia
paolo.ciancarini@unibo.it

Sergey Masyagin
Innopolis University, Russia
s.masyagin@innopolis.ru

Giancarlo Succi
Innopolis University, Russia
g.succi@innopolis.ru

Abstract

Are we really *writing software*? What do software writers have in common with other professional writers? What can we software developers learn from professional writers? This paper proposes a reflection on such topics using as a reference the book "*Six Memos for the Next Millennium*", a posthumous essay by the Italian novelist, editor, and literary critic Italo Calvino. A comparison is drawn between such work and the current principles ruling how software should be written and developed, and a claim is made that this is an area worth further exploration.

CCS Concepts: • **Software and its engineering → Software design engineering**; *Requirements analysis.*

Keywords: Software design, System metaphors, Quality Attributes, Common Practices, Software Professionals, Natural language analysis of source code artifacts.

ACM Reference Format:
Paolo Ciancarini, Sergey Masyagin, and Giancarlo Succi. 2020. Software Design as Story Telling: Reflecting on the Work of Italo Calvino. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '20), November 18–20, 2020, Virtual, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3426428.3426925

1 Introduction

More or less since its early days, software development has been considered an engineering discipline, see for instance [19, 63]. However, this has always been a controversial issue, see for instance the long legal proceedings that arose when Memorial University of Newfoundland positioned a degree in Software Engineering in a Faculty of Science[3]. Furthermore, "traditional" professional engineers have cast doubts on the "integrity" of the process followed in software writing and on the "legality" of calling engineer a person who does not execute some hard core construction (whether of a building, of a mechanical device, or of something anyway physical).

Software design is more and more a practice needing a plot, some characters and a narrative point of view. We undervalue storytelling and focus too much on modeling, coding, and libraries. Reading a piece of code, modeling a system architecture, or depicting a process workflow can be very frustrating, without a narrative. It is the story, the rationale left behind by that code, architecture or workflow, that is just as important as the artifact itself.

Software development has been recognized as an activity in which technical issues are strongly intermingled with social issues. In fact, software development is about reading and writing code, which is by its nature a socially creative activity [28]. Programs are not just aimed at computers to run: they are texts shared among software engineers and developers, as software systems are typically not written by individuals alone but by teams of people. Moreover, they are also a vehicle for maintenance, since often a program is poorly or unreliably documented, so its code becomes the best resource for understanding what (a portion of) a system is about. Moreover, individual pieces of code can become vehicles for knowledge sharing. Altogether, designing and writing code is effectively a communication act assuming a readership and some storytelling, like an essay, a story, a poem…[49].

Needless to say, the code has to be written and documented in a way that is simple for people to read and understand it. Well, there is not much novelty in this statement taken alone, since from the very first days of software engineering researchers have studied how to write programs that could be clearly understood. A good portion of the work done in empirical software engineering and in software metrics is exactly on trying to determine how to organize the code so that it is more readable, has a lower number of defects, is easier to maintain, etc. [6, 72].

In a different domain, there are quite ancient disciplines giving suggestions and prescriptions on how to write, including first of all *Poetics*, named so after the work of Aristoteles [1], and also *Rhetoric* and *Aesthetics*.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Onward! '20, November 18–20, 2020, Virtual, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8178-9/20/11…$15.00
https://doi.org/10.1145/3426428.3426925

Our claim in this paper is that these disciplines have not been analysed enough in a software development perspective, even if they have a large relevance on to how software is described and designed and written as a code, a relevance that is made evident by the frequent analogies existing between observations coming from poets and writers and the recommendations that have come throughout the years from software engineering scholars and practitioners. An example is [66].

A notable example of such analogies are the so-called three Aristotelian unities on how a theatrical play should be structured. To be precise, even if these units refer to Aristoteles in their names and come from the mentioned Poetics, their current formulation is the result of a re-elaboration of the work of several scholars during the 16th century, among them Lodovico Castelvetro (1505-1571) [23]. The three unities prescribe that a play should be organized in terms of:

- unity of time, meaning that the facts presented in the play should span a limited period of time, typically one day, even if it may refer or recall other facts occurred before or after using flashbacks and flashforwards;
- unity of place, meaning that the facts presented in the play should occur in a limited location, usually one city, again, even if they may relate to other locations using specific rhetoric figures;
- unity of action, meaning that the facts presented in the play should refer to well defined actions, usually one single situation, again, even if they may relate to other situations

A way of interpreting these principles is that, when writing a play, narrative dependencies should be minimized, avoiding to complicate the plot in time, space, and actions. Hence, these three units can be easily referred to principles on how to write code, here below there are some notable examples:

- time: a properly written function should have a low fan-in and fan-out, that is, should limit its dependencies on other pieces of code, to global variables, etc.
- place: a properly written function should span no more than one page to be readable, understandable, modifiable, reusable, etc.
- action: a properly written function should refer to at most one functionality of the code to be cohesive and reusable.

We can find many similar analogies. We can also draw correspondences between software development and other artistic activities, like music and visual arts. On these subjects we have entertained various conversations with colleagues, so we prefer not to deepen the discussion to prevent ideas that were not originated by us to emerge as ours, but it is just worth mentioning that the symmetry between the improvisations of Bach and agile software development or between the sketches of Leonardo da Vinci and prototypes are more than interesting coincidences.

Still, the thesis of this paper is that there is much more than a pure analogy linking principles of poetics to principles of software development, because they share a very similar kind of creative process and the same medium of communication, to the point that there should be a poetics of software development, and we think that not only we can, after the fact, relate principles of software development to various principles discovered in poetics, but we can also proceed in the opposite direction, that it, studying the poetics to understand why certain pieces of software appear "better written" and thus discover better, more efficient, and effective ways to write software.

To prove this thesis we focus our analysis on a work by Italo Calvino, an Italian writer born in Cuba, culturally linked to both the East and the West of the world, having worked in Moscow and New York, and many other places around the world. Specifically, we analyse his posthumous work "Six Memos for the Next Millennium", lectures prepared for Harvard University, published both in Italian [21] and in English [22] almost at the same time. This work was intended to be a collection of six lectures on what he thought the third millennium should learn from the second, especially from the 20th century. The lectures were to be given at Harvard University as part of the "Poetry Lectures" in 1985, but they never took place because of Calvino's death. The Lectures have inspired some philosophical analysis in mathematics, as in [50, 51].

In this work we analyse the five fully completed lectures, which concern the following five properties of literary works[1]:

- Lightness
- Rapidity
- Precision
- Visibility
- Multiplicity

We will not discuss the sixth lecture on Consistency, since it was not fully elaborated by the author.

Considering the above list of terms, we observe that some of these resonate with terms from agile software development. In this discussion we discuss this analogy, aiming at emphasizing what is present in Calvino's vision that is not present (yet) in current software engineering practices.

Altogether, the goal of this paper is twofold:

- On one side, to promote further and deeper reflections on the analogies between writing software and various creative arts, first of all, writing. Now we are mostly doing a retrospective, analysing how software engineers have drown from their thoughts and their

[1] In the discussion we depart from the typical English translation of the work of Calvino, which uses "quickness" and "exactitude" instead of respectively "rapidity" and "precision" because we consider the latter terms more evocative for software writers.

experience similar conclusions as art critics; however, the ambitious aim would be to improve our discipline being able not only to do a retrospective thinking but proactive proposals of changes based on reflections and comparison with such very large and eminent body of knowledge.
- On the other side, use Calvino as a case study for our thesis and also as a concrete reference.

The remaining of this paper is organized as follow. The next section 2 introduces Calvino's work and recalls some issues concerning agile developments and story telling. Sections 3 to 7 present the five mentioned qualities, first as they were discussed by Calvino, then contrasting them with current software development principles and practices. Section 8 discusses our findings, and Section 9 draws our conclusions and outlines possible avenues for further investigations.

2 Background

2.1 Calvino

Italo Calvino was born in 1923 and died in 1985. He was a novelist and essayist. He was especially the former, in fact he is still considered a master story teller, the most important Italian story teller of the last century. As a novelist he wrote among others the novels *Cosmicomics* (1965), *Invisible Cities* (1972), and *If on a winter's night a traveler* (1979). As an essayist he wrote among others *Six memos for the next millennium*, a book that he left unfinished, published posthumus (1988) and that we will discuss here in this paper.

Our interest in Calvino's work stems from some intuitions he had about computing, that he exposed in his book "Six memos..."; the most well known is the following:

> *...then, Computing. It is true that software could not exercise the power of its lightness except through the heaviness of the hardware; but it is the software that commands, that acts on the external world and on the machines, which exist only to support the software, and evolve in order to elaborate increasingly complex programs. The second industrial revolution does not appear as the first with overwhelming images such as rolling mill presses or steel castings, but as the bits of a flow of information that runs on the circuits in the form of electronic impulses. The iron machines are always there, but they obey the weightless bits. "*

And again, reflecting on lightness, he anticipated the pervasiveness of the digital world:

> *When the human kingdom seems condemned to heaviness, as in the current times, I think I should fly like Perseus in another space. I'm not talking about escapes into dreams or the irrational. I mean I have to change my approach. I have to look at the world from another perspective, another logic, other methods of knowledge and verification. The images of lightness that I am looking for must not allow themselves to dissolve like dreams from the reality of the present and the future ...*

The book discusses the following six concepts related to story telling:

1) Lightness: meaning lightness of touch, rather than lack of seriousness: this quality helps both writers and readers who practice and search for lightness to improve their ability to know and operate the world. 2) Rapidity: the mental, essential speed of a story told to entertain. The story should pull the reader along and not get mired up in questioning the non-essential parts. 3) Precision: the novel should be structurally proportioned. Calvino reveals that his guiding image when composing a literary work is the crystal âĂŤ its complexity and the fact that it can be held in one hand and admired despite all that complexity. 4) Visibility: the visual nature of the literary work is also important. Every story begins as a visual cue, to which more and more images are added until he has to summon words to describe this profusion of images. He worries about what will happen to the originality of the visual imagination in a world saturated by inessential images. 5) Multiplicity: a literary work should try to encompass the world as a network of relationships. Every literary object is related to an infinite set of other objects, and the writer should be able to master and exploit these infinite relationships. 6) Consistency: the lecture concerning this concept was never written.

Each lecture can be linked to some of the author's novels: for example, the theme of lightness is dealt with in The rampant baron; The path of the spider's nests is instead linked to rapidity. Proceeding in this way, one can connect the precision to Cosmicomics and to Ti with zero; Visibility to Invisible cities, Multiplicity to If a traveler on a winter's night.

2.2 Agile Software Development

In this paper we discuss software design as story telling according to Calvino from an agile perspective. We assume that the Agile Manifesto is well known among our readers, who are also software developers or software process scholars. In short, the Agile movement puts emphasis on people interactions, on working code, on conversations among stakeholders, and on short iterations planned each to release a working prototype useful for the stakeholders.

The idea of exploiting story telling in Agile is in some sense natural, however it is not usually related to literary story telling. We have found only one report dealing with this issue [25].

Most agile methods are based on user requirements described as short stories - these are called user stories but have a specific format and are restricted to one sentence only.

Another point of contact of agile design with story telling is the concept of system (or architectural) metaphor. According to K.Beck: "The system metaphor is a story that everyone–customers, programmers, and managers–can tell about how the system works." [10].

We will discuss these and other issues in the rest of this paper.

3 Lightness

The first lecture starts with the feelings that Calvino had when he started to write.

When I started my activity, the duty to represent our time was the categorical imperative of every young writer. Armed with good will, I was trying to identify myself with the merciless energy that is moving the history of our century, its collective and individual events. I was trying to capture the harmony existing between the colourful show of the world, sometimes dramatic, sometime grotesque, and the picaresque and adventurous interior rhythm that was driving my writing. Soon I realized that between the fact of the life that should have been my raw material and the dynamic agility that I wanted to animate my writing there was a gap that was always more difficult to overcome. Perhaps, then I was discovering the heaviness, the inertia, the opacity of the world: qualities that get immediately attached to writing, if we do not find the way to escape from them.

This text emphasizes the two contradicting aims of writers:

- to provide a comprehensive description of the world, and
- to be agile in writing, easy to read, lively, engaging.

3.1 Lightness in the Poetics of Calvino

It is particularly revealing how Calvino handles the concept of *heaviness*. Rather than blaming it, he uses a metaphor. Complexity is like Medusa, who attracts inexorably anyone trying to contemplate her. Therefore, Perseus kills her cutting her head looking at a reflection of it on a kind of mirror, but after doing it, he does not throw the head away. Rather, he keeps it with him to use it as a weapon against his enemies, and to this goal puts it in a nice and honorable container, like a sword in a scabbard.

The heavy burden of complexity is not to be ignored. First, it has to be tamed, but since it is impossible to do it directly, the taming occurs through mediation and variable perspectives. Then, once tamed, it has to be always taken care of and never forgotten. The winning approach is to understand slowly, acknowledging our limits, which make impossible an upfront comprehension, and then to take it always with us, showing it any time someone proposes us to forget about it.

I hope to have demonstrated that there is a lightness of thoughtfulness and a lightness of levity, and that the lightness of thoughtfulness can make the lightness of levity to appear heavy.

Said differently, we often "linearize" problems, but then we are exposed to the risk of forgetting such linearization and of treating the linear model as the unique reality, which is not the case.

The second issue follows from using lightness to express the constant mutability of the reality. Calvino notices that there are multiple writers that describe a universe in constant evolution and that the only way to do it in an , which we could call in software the evolvability of systems.

3.2 Lightness in Writing Software

Needless to say that the reflections that Calvino on his beginning as a writer resemble strongly the situation people faced when the agile movement arrived, and so we can almost claim that the overall description of lightness actually corresponds to what software engineers defined a few years later as "agility", a term that is also used by Calvino in his narration. Lightness can also be mapped in other terms in software, including the prescription to play down what we develop, to make cohesive classes with limited functionality, to minimize dependencies, etc.

In particular, it is revealing the difference that was made between:

- the approach of the big upfront design aiming at a full and comprehensive description of the reality composed of reusable entities like objects, claiming that this development strategy would have carried the most effective approach to software development [65];
- the approach of the incremental design, which consists in trying to learn the reality and solve problems while learning with a fast and cutting agility [10, 27].

The allegory of Perseus is also quite revealing. Just trying to be overly light ends up in creating a heavy chaos. This concept has also appeared somehow in some agile terms. It has been overly rediscovered and repeated, for instance by Van Gogh "How difficult it is to be simple!"[2] and in software by most agilists, including Beck [9]. But here there is more and something we have not yet been able to capture, that is, the explicit acknowledgment of the need to perceive the complexity, indeed perceive it like Perseus, through simple images, but to handle it. In a sense, these was approached by trying to create simplified views of complex structure, for instance via tools identifying variability links, but still it is something not fully exploited. And going to the issue of "linearizing problems," when we code, we write light classes, but then we may forget the complexity we have decided to ignore, the features that are cut, and so on.

[2]Letter from Vincent van Gogh to Paul Gauguin, 17 June 1890

3.3 Additional Lessons to Learn from the Reflections of Calvino on Lightness

The description of lightness triggers several reflections on how we write software, in particular with respect to aspects of the lightness that may shed new ideas to software development especially for what it concerns lightness as deprivation or lack of resources. In this sense Calvino reflect on Kafka and says:

> *The empty bucket, sign of deprivation, desire, and quest, which elevates you to a point where your humble prayer cannot any more be answered, paves the way for endless reflections.*

The concept of deprivation to understand what cannot be written is very deep and not fully understood in writing software. It is like using an incomplete language and then taking advantage of the incompleteness to hint implicitly at what it is impossible to express explicitly due to such incompleteness, and also at much more. It could be an effective way to manage variability.

And, indeed, lightness a deprivation can be a tool to handle mutability and variability, exactly like in literature.

4 Rapidity

The second lecture focuses on rapidity. Calvino emphasizes that it is not a fast random movement but the result of competence, knowledge, and thinking. He refers to rapidity with the Latin aphorism *"Festina lente!"*, that is, "Be fast, slowly!" He then describes it with the help of the following tale.

> *Among the multiple virtues of Chuang-Tzu there was the ability to draw. The king asked him to draw a crab. Chuang-Tzu replied that he needed 5 years and a villa with 12 servants. After five years the drawing was not yet started: "I need five more years" said Chuang-Tzu and the king accepted. At the end of the ten years Chuang-Tzu took the brush and with a single movement draw a crab, the most perfect crab ever seen.*

4.1 Rapidity in the Poetics of Calvino

Dealing with rapidity Calvino clarifies that in the creation of a poetics, aiming at achieving (in part) a quality is not contradictory with achieving its opposite. He refers to a psychoanalytic analysis of two Gods of the ancient Greek and Roman mythology:

> *Mercury and Vulcan represent two inseparable and complementary vital functions: Mercury the syntony, that is, the participation to the world around us, and Vulcan the focus, the constructive concentration. Mercury and Vulcan are both the sons of Jupiter, whose kingdom is that of the individual and social conscience. However, from the maternal side Mercury descends from Uranus, whose kingdom was the continuously undifferentiated "cyclothymic" time. Vulcan descends from Saturn, whose kingdom was of "schizophrenic" time of the egocentric insulation. Saturn has overthrown Uran, and then Jupiter has overthrown Saturn himself. At the end in the harmonious and enlightened kingdom of Jupiter Mercury and Vulcan bring each their memory of the dark primordial kingdoms, transforming what was a destructive illness in positive qualities: the syntony and focus.*

So over the concept of rapidity Calvino reasons in two ways:

- rapidity as the result of a profound mastery of the discipline and a deep understanding of the problem to face – the story of Chuang-Tzu, who "with a single movement draws a crab, the most perfect crab ever seen," but only after 10 years of meditation and preparation;
- rapidity as one of the two extremes - non mutually exclusive - that are present in the minds and in the behaviors of people, that need to be balanced properly to have an harmonious life.

Taking into consideration these two fundamental views of rapidity, we can now expand some specific ideas of Calvino's about the role of rapidity.

Calvino emphasises that using concrete objects promotes rapidity in narration:

> *Around the magic object there is a force field that is the narration. We can say that the magic object makes explicit the connection between people and events. . . . And in a narration an object is always a magic object.*

Calvino goes on saying that every object has properties and these properties help to increase the understanding of the narration.

The second concept is the importance of the oral narration, as exemplified by the folktales and fairy tales that are passed by oral tradition.

> *If during a time of my work in literature I have been attracted by folktales and fairy tales, it has not been for loyalty to an ethnic tradition (since my roots are in a modern and cosmopolitan Italy), neither for nostalgia for child stories (in my family a child had to read only educational books with some scientific background), but for stylistic and structural interest, for the economy, the rhythm, the essential logic used to tell them.*

Lastly, Calvino evidences that it is difficult to keep alive the attention of people writing long stories, so he prefers to express concepts in short stories, if needed, related one another.

4.2 Rapidity in Writing Software

The concept of rapidity, as explained by Calvino, has a significant importance also in software.

First of all, it is important to evidence that also in software engineering it is not uncommon to aim at achieving sometimes apparently opposite goals; often, in reality such opposite goals are not creating a conflict, like in literature, but represent different views of what to achieve. This boils

down to the need to proper contextualizing goals, as it has been evidenced strongly by the groundbreaking approach for setting goals in requirements engineeering [70] and in the GQM (Goal-Question-Metrics) framework [7]

The double view of rapidity is also largely present in software: on one side, as mentioned multiple times by some agilists like Kent Beck [8, 10, 11, 29], agile process models, the most prominent class of "rapid" development processes, require a deep mastery of the art of software development.

On the other side, there is not any single methodology that fits all situations, like there it not a unique poetics for all sorts of creative composition. Furthermore, to develop properly software, it is essential to properly mix rapidity, as the art of understanding fast, and being in an harmonious "syntony" with the customer and her/his needs, the development team, the operating environment, the technology to use, etc., and slowness in the sense of capturing the deep of the need and to gather the knowledge that is required to carry our specific projects properly. This has been discussed, even if in a simplistic way, in the works by Boehm and Turner [15–17] and of many others [13, 24, 48, 68].

The role of the "object" as the center of narration and of code writing is very clear when dealing with object oriented systems, and it is also true that objects contain a sort of "power" to attract the attention of developers, with also the risk of creating a monstrous amount of code around them, as it has already been described several times [62].

Furthermore, oral narration is a direct contact between the author (or the teller, who often becomes a re-writer) of a novel and the readers who become listeners or even spectators. This is indeed what the Agile Manifesto refers to with "Individuals and interactions over processes and tools" [12]. The art of communicating with the customer directly, concisely, and in a non ambiguous way is indeed a key asset of the agile approach.

Finally, the concept of describing the reality in terms of small stories rather than long descriptions is typical also of agile methods, for instance in the user stories.

4.3 Additional Lessons to Learn from Calvino's Reflections on Rapidity

Calvino discusses the importance of knowing how to stop when telling a story, that is the importance to omits parts of the description, which is not just hiding the details. Information hiding is a cornerstone of software design [58], however, it represents only one side of the omission of description. Calvino stresses that this omission of details is instrumental for interconnecting different parts of a narration without loosing focus, proceeding with "rhythm." And rhythm promotes understanding – Galileo said that *argumenting is like running*. This aspect exhibits also potentials for application to software; it is already present inside agile methods, when referring to a constant pace of work, for instance, the 40 hours per week recommended by XP [10] ; however, its application not only to the process but also to the code has further potentials to be explored, especially with reference to the idea of expressing concepts by omitting descriptions.

Calvino also emphasizes the concept of repetition. This is also not yet properly covered in describing software development even if all the work on reuse, especially that on design patterns, has strong similarities to it [36].

5 Precision

The third Calvino's lecture is about precision, whose meaning is immediately defined as:

- a well defined and well planned design of a work;
- a recollection of images that are clean, crisp, well defined, easy to memorize, vivid;
- a maximally precise language, both in terms of usage of the words and of ability to render thoughts and imaginations.

5.1 Precision in the Poetics of Calvino

Calvino claims that it is as if the humanity has been plagued now by an epidemic of lack of precision. For defining *precision* we use directly Calvinio's words, as they appear particularly appropriate:

> *Often it appears that a pestilential epidemic has plagued the humankind in the aspect that characterizes it the most: the use of the word, a pest of the language that manifests itself as loss of cognitive strength and of spontaneity, as an automation of forms that tend to level expressions toward more generic, anonymous, and abstract formulations, to smooth down crisp expressions, to turn off any sparkle arising when old words occurs in new circumstances. I do not care here whether the root of this epidemic are in politics, in ideology, in the bureaucratic uniformity, in the homogenisation of mass-media, in the scholastic diffusion of middle education. What I am caring about are the possibility of healing. The literature (and perhaps only the literature) can create the antibodies that may contrast such pest of the language.*

Furthermore, Calvino argues strongly that precision is not completeness, emphasizing that his writing has to satisfy two, apparently divergent constraints:

- on one side, the adherence to a model based on the story to narrate,
- on the other side, the ability to express in limited space such complexity, without compromising the precision.

He stresses that the solution for this dilemma is not to "fill pages with words" in a humongous effort to put in places all such details, rather to separate what is expressed with words and what is expressed with omissions.

5.2 Precision in Writing Software

Reading this text, immediately we think at how often nowadays we encounter code that is quite unjustified and imprecise, sometimes copied from the web just because "it works" without a full understanding of its meaning, causing scores of problems in the future. People have often referred to this problems a "technical debt" [47], and in this context we can assume Calvino as a major supporter of the elimination the technical debt.

Moreover, Calvino's reference to the epidemic of coarse approximations recalls the current situation where some software engineers take shortcut solutions instead of understanding in deep the problems they face. We refer to our anecdotal experience of some senior professionals claiming to know Android, simply because they could write apps using an IDE, but still not understanding, for instance, that layouts are objects created by reflection through dependency injection [54], or self promoted data scientists able to create models through Kera but unable to explain with proper argumentation the architecture of the network and the values of the associated hyperparameters [34]. And this not to mention the even worse situation when professionals claim to be expert of methodologies, like agile methods, without any thorough understanding of the corresponding discipline, just replicating as parrots terms coming from – one of the authors discussed this phenomena extensively in previous works [39, 40].

Altogether, it appears that Calvino's quest for precision reflects closely the quest for precise design of proponents of agile methods, especially when they strongly assert the need for simplicity, refactoring, technical excellence, and team and personal reflections. These four components reflect the Calvino's quest for precision and for a language that becomes pure, and we would not be surprised in reading an addendum of the Agile Manifesto claiming exactly that "Agility (and perhaps only agility) can create the antibodies that may contrast such pest of the language."

We have to reflect on the fact that probably the last large-scale controversial discussion in software engineering on comparing the features of programming languages has occurred decades ago around Aspect Oriented Programming [46]. Such discussion did not raise in any way flames similar to the work by Dijkstra on "goto" [31], by Backus on assignment and functional languages [2], by Turner on indentations [69], or by Kay on Smalltalk [43]. This does not mean that using tools is bad or that tools reduce knowledge. However, tools and frameworks should not become proxies for knowledge; they are excellent mechanisms for fast prototyping or for less educated people to perform pedantic work, but this is it, and should be very clear.

Finally, the battle between precision and completeness is a constant struggle when writing software, when features are added to data structures, to objects, in a quest for precision. This is indeed always the struggle of software engineers, between the quest for an absolute and abstract formalization and the need to express the crucial details of the reality, which often resist to formalization.

5.3 Additional Lessons to Learn from the Reflections of Calvino on Precision

Calvino makes an important point on a special form of imprecision, the vagueness. He claims that vagueness has a very positive connotation for many poets because it allows to perceive with a high level of precision (notice the oxymoron) the indetermination of several situations. He refers to the Italian poet and writer Giacomo Leopardi, who in his *Zibaldone of Thoughts* on the date of 28th September 1821 annotated: "To this pleasure contributes the variety, the uncertainty, the fact that we cannot see everything, and so we can elaborate with our imagination what we cannot see."

Writing programs we have often touched such issue, but we have never had the courage to accept that a lack of precision can trigger a better understanding. However, let us consider a simple example: if we have an index of a loop, is it more understandable to call it `i` or `indexOfALoop`? The latter is indeed a more precise approach than the former – there have been even empirical studies evidencing this [14]. Overloading names is also indeed an act of vagueness and it forces the mind to contextualize the term – however, indeed, it triggers higher understanding. Not to mention that specifying only what is strictly needed and deferring commitments is a key tenet of agile. An appropriate use of vagueness appears therefore a strong mechanism to increase understandability and adaptability of code.

A related concept is the appreciation of the impossibility of generalizing: there are situations that appear to be absolutely unique. Calvino cites Robert Musil in *Der Mann ohne Eigenschaften* (1943): "There are mathematical problems for which a general solution does not exist but rather individual solutions, that, together, approach the general solution."

Regardless of the truth of Musil's statement, Calvino emphasizes the importance of being specific, of avoiding the temptation of generalizing and abstracting. He cites Flaubert "The good Lord is in the details," and connects him with the cosmology of Giordano Bruno, who imagined the universe unlimited but not infinite.

This aim is somehow present in some aspects of agile methods. Still, raising such ideas to an overarching principle of "locality" is totally new. This amounts to saying that there are situations where not only generalizing is not profitable and results in a waste of resources, but also is a mistake and leads to wrong solutions. Calvino elaborates this statement further:

> *In reality, always my writing has found itself in front of two divergent ways that correspond to two different kinds of knowledge: one that moves itself in the mental space of*

unbundled rationality, where we can draw lines that interconnect points, projections, abstract forms, vector of forces; the other that moves itself in a space full of objects and tries to create an equivalent verbal of the space filling pages with words . . . They are two different compulsions toward precision that will never arrive to the absolute satisfaction . . . I oscillate continuously between these two ways and when I feel to have completely explored the possibility of one I jump to the other, and viceversa.

Then, Calvino goes ahead mentioning that the poet should describe one facet of a crystal representing the reality, where every side of the crystal provides a different and complementary views of it. He mentions several authors that followed this approach, like Paul Valéry, Wallace Stevens, Gottfried Benn, Fernando Pessoa, Ramòn Gòmez de la Serna, Massimo Bontempelli, and Jorge Luis Borges.

This is exactly what is important in software development, to write code covering individual aspects of a problem, but still being able to be integrated in a whole, and this without describing the whole.

An attempt on this was done in the past, with the concepts of modularity and abstract data types. Still such approaches require a vision of the whole to write the individual, and they pose constraints in the vision of the whole. The metaphor of the crystal, on converse, support the idea of writing of the whole, without putting concerns on the whole.

6 Visibility

Visibility for Calvino is not simply a property of entities that can be perceived by our eyes, that is, that can be viewed. Rather, it is the process by which the fantasy elaborates images of sensations and feelings.

To explain the concept of visibility, Calvino refers to Dantes's verse of the 17th *Canto* of the Purgatory: "It rained inside the high fantasy." Here there is an emphasis that visibility is the process of acquiring information from sensations. Calvino observes that Dante mentions "high fantasy" not just "fantasy." In Aristotelian terms, we are considering the highest level of imagination, the one which gets purified from all possible accidental events. In a word, it is a very "simple" vision; in this framework, therefore, there is a demand for simplicity. Vision is important because is pure, and purity requires the elimination of everything that is superfluous.

Software is invisible but our imagination makes it visible via diagrams, metrics, all mental elaborations where the ideas "rain inside our fantasy" and become concrete. Such diagrams can be wildly different, as a product of the imagination of different people; consider as an example Object Oriented systems: there have been many proposals of strongly different diagrams, before UML took over [37].

Likewise, the poetry handles invisible ideas and feelings and makes them visible. Calvino links visibility to imagination and fantasy, that are able to create ideal images that capture essential but yet difficult parts of the reality.

Therefore, all the following discussion refers not just to the abstract concept of visibility, but to how such concept triggers clear images of the reality and stimulates the fantasy to capture even deeper how our world is structured.

6.1 Visibiliy in the Poetics of Calvino

To explain his view on Poetics, Calvino refers back to the 17th *Canto* of the Purgatory: "Oh imagination, you who capture our attention so strongly that even 1000 trumpets could not distract us from you, who move moves you even when what our senses do not instruct you?" In the view of Dante (and of Calvino) this abstraction process is so strong that it is able to create new solid models of the reality that go beyond what we can simply sense or perceive.

Calvino clarifies that visibility plays a dual role in the overall understanding process:

- the first is when we provide a visual representation of the reality, so we go from discussing or reading to images or diagrams;
- the second is when we see an image and we reconstruct the reality in a verbal form, like when we discuss over diagrams that we have analysed.

Then, Calvino refers to the role of fantasy for mystics and ascetic quoting Ignatius of Loyola, who recommended during prayers to build concrete images of Christ. This is indeed an essential component of software development; which goes beyond the simple role of building abstractions. The ability to create images of the running software is a pillar of agile methods; sometimes we refer to this with the

The next important point that Calvino targets is how the imagination and visual models are generated. Going back to the experiences of Jean Starobinski in his work "The Empire of Imaginary" (from the essay "The Critical Relationship" [67]), and also drawing from Freud and Jung, he discusses how images gets generated and presents his approach in writing novels. Here he outlines that actually he starts from an image and then elaborates from the image writing down a story, and this story then suggests another image. Therefore, the narration is:

- a tool to understand,
- and a way to provide additional insights and ask for an additional understanding,
- but importantly also as a collection of possible hypotheses of what could be.

Then Calvino makes a prophecy about the evolution of images, as he foresees:

- on one side, the standardization of the images that are being

- on the other side, a complete restart of the image creation process, especially for new and more challenging tasks.

6.2 Visibility in Writing Software

The overall idea of how the imagination works according to Dante and Calvino define a clear essential guideline in coding, to create simple and powerful abstractions able to generate new ideas. As in Calvino, in software development practically there is a general awareness that images have a superior ability to represent the reality than simple words.

Moreover, what Calvino refers to as the dual role of visibility in understanding occurs in coding and in software development. This is exactly the the whole work on representing software architectures or models via sets of diagrams and then move from the diagrams down to written descriptions as in PlantUML is a clear example of this [52, 56], so to make diagrams accessible to "blind people". These two directions have taken sometimes the name of "roundtrip engineering" [30].

With respect to the example of Ignatius of Loyola, we know that visibility facilitates agility through the creation of vivid images of what we are developing. System metaphors in XP are an example: the XP practice of metaphors is strongly connected to user stories; these are simple enough to make them understandable for the customer and final users, at the same time they fully describe the functionality of the software product [45].

Furthermore, visibility helps to promote continuous improvement. By openly sharing successes and failures the teams, and thus their organization, can steadily evolve how they work which will help to promote individual growth and the creation of better technical solutions. Visibility also via Dashboard, on a continuous basis, enables an organization to make key adjustments based on customer or market feedback, technology changes, and more.

Through collaboration, with both internal and external team members and stakeholders, agile developers can exploit the opportunities to inspect their process and make adjustments to improve efficiency and productivity. Process inspection occurs daily beginning with the "daily scrum" (or standup) and continues through to the sprint review and retrospective. Each day, team members and other stakeholders will inspect project indicators, such as burndown charts, cumulative flow diagrams, and other metrics. This visibility help to expose process issues as they occur and allow for swift adjustment.

With respect to how imagination and visual models are created, we can see that there is a clear iterative process that resembles quite strongly the agile process of writing user stories elaborating scenarios and then creating other user stories. Moreover, using the narration as a major tool for discovering what need to develop is fully coherent with the incremental development typical of agile methods, where the requirements are better understood and explored further while the system is being developed and released to customers, incorporating the feedback from the customers on the new development. Altogether, we can say that quite like narration, the agile way to develop software incrementally is (text in italics is copied verbatim from above):

- *a tool to understand*,
- *and a way to provide additional insights and ask for an additional understanding*, for instance for the customer,
- *but importantly also as a collection of possible hypotheses of what could be* further developed.

Needless to say, user stories are very effective means toward this goal.

Interestingly, also the evolution of images reflects the situation in software engineering.

- *the standardization of the images* has been apparent on the standardisation of user interfaces and of icons that has taken places for handheld and mobile devices in the last 20 years
- *a complete restart of the image creation process* is also clearly present in all the field of dashboarding and business analytics.

6.3 Additional Lessons to Learn from the Reflections of Calvino on Visibility

Additionally, the ideas of Calvino on visibility provide us additional insight on how to write software. Referring back to how images are generated, Calvino outlines two possible interpretation of the process of building representations of the world:

- as communication of internal state of minds
- as emergence of knowledge archetypes

In other terms, with the second approach it is as if there are a set of predefined skeletons of models present in the mind that emerge to form the visual images of the reality.

All these concepts almost completely new in software engineering. There are limited studies on how images get formed and the latter approach is like taking one step up the concept of design patterns. However, a better understanding of how images are formed would significantly shed new and interesting lights on how to write efficient software. More than something to learn from poetic, is a research endeavour to copy from them.

Linked to this, Calvino reviews of the "physical" exercises to stimulate the creation of the visual images. Referring to Ignatius of Loyola, he describes how specific exercises can help promoting visual images. This is not anything new in meditation, where people learn to meditate always better via regular exercises [71], but it is definitely something unseen in literature before, and definitely not present in software engineering. Calvino promotes a pedagogy of imagination, that foster the creation of abstract models but also helps avoiding the random production of unordered pieces of elements.

7 Multiplicity

According to Calvino the concept of multiplicity is at the root of literature and arts. He writes:

> *Excessive ambitions can be blamed in many areas of human endeavours, but not in literature. The literature lives only if it sets humongous objective's, even beyond what is conceivable. Only if poets and writers will aim at achieving enterprises that none else even dare to dream the literature will continue to have a purpose. At the time at which the science is wary of general explanations and of solutions that are not sectorial and specialized, the big challenge for literature is to be able to weave together the different knowledge and the different codes within a plural and multifaceted view of the world.*

Multiplicity is inherent in software: multiple authors, multiple languages, multiple platforms, multiple - possibly infinite - interactions. Moreover, multiplicity brings together a core issue common in software development and in poetry: the level of granularity needed in a representation. The problem is how many details of the world are needed to describe a situation or scenario and how such details refer or link to other situations or scenarios.

7.1 Multiplicity in the Poetics of Calvino

In the description of how multiplicity has been handled by story tellers, Calvino identifies two extremes, but without being able to define what is for sure the best approach:

- on one side there is a tendency of provide a plenitude of details (like Carlo Emilio Gadda) and
- on the other side, there is an approach to minimize the possible information flow (as it is done, for instance, by Robert Musil).

Calvino, then, evidences that a non trivial number of writers across centuries and languages have the tendency of not concluding their work. Just to give an idea we can list Virgilio (Latin, 70BC-20BC), Dante (Italian, 1265-1321), Goethe (German, 1748-1832), Proust (French, 1871-1922), and many others. Such tendency of not concluding and of aiming at always better and more accurate representations of the reality or a higher quality of the writeup refers back to the issue of precision, which we discussed in Section 5; Moreover, it is clearly linked to the fact that it is hard to define a stopping criteria when writing a book and it is always possible to enrich a volume or a history also after its apparent completion.

This fact is well exploited also in a writer that Calvino could not have known, Joanne Rowling. If we notice the history of Harry Potter started and then spanned several books, probably well beyond the original desire of the writer, and went even further elaborating corollary works around her original idea, taking advantage of the success that she had.

Finally, Calvino organises the management of multiplicity in literature in four classes:

1. texts following a *single inspiration, but that can then be understood at multiple levels*; already Dante in the outset of the 14th century wrote in the second chapter of his book "Convivio" an essay outlining 4 levels at which to interpret a poetry; however, such essay stayed predominantly within a theological framework. Here Calvino goes beyond such boundaries and refers to the work of surrealistic writers, such as Alfred Jarry;
2. works where *multiple narrations* intersect strongly within a well defined context, but without providing an overall unifying vision; this is like the Comedy of Art of the 16th and 17th century – the structure of comedies before the unifying revolution lead by Carlo Goldoni, where different masks like Arlecchino, Pulcinella, and other interacted in a general play with coherent action but without a strong line connecting the beginning to the end;
3. attempts to create a *single and unifying vision* of the whole work, that then is very likely to never be completed; we have already evidenced the numbers of authors that, across centuries, were not able to complete their work;
4. collections of *non contradictory aphorisms*, which do not give a comprehensive and explicit view of the subject of discussion, but shed lights on aspects of it, aiming at finding the simple and small cores of narration; to clarify this Calvino cites a writer of such short booklets of aphorisms, Valéry, saying "I have looked, I am looking, and I will be looking for what I call the *Total Phenomenon*, that is the essence of conscience, relations, conditions, possibilities, and impossibilities."

7.2 Multiplicity in Writing Software

Multiplicity is a core problem in software writing, sometimes referred to as "variability" [59]. In this context the two extreme approaches described by Calvino are well represented also in software development:

- there are methods that attempts to create comprehensive models of the systems to develop, presenting all the conceivable details, and such approaches were underneath, even if not necessarily made explicit, not only the traditional waterfall methods but also object orientation during its infancy, like for instance in [5] and in [18];
- on the contrary, there are alternative ideas that software should be minimalistically designed, focusing only on the minimal aspects providing the essential required functionality to the users, as clearly stated in all the works related to agile methods, like the ones of Kent Beck [9]

After the initial enthusiasm for agile methods, there is now the understanding that is not clear toward which end should software development lean. Typically some methodologists

suggest heuristics on where to lean avoiding the (perhaps impossible) answer to the overall question [15]; such heuristics are often so obvious that they imply that there are not clear criteria on what to do, and that the software engineers in charge of the project should base the decisions on their own experience, indeed, at their own risk.

Likewise, especially in the time of the waterfalls, software development has the natural tendency of never ending, or to end only when the budget expired – like books ending only when the author died. Also in software there is a risk that endeavours never conclude, as it is widely documented both in scientific documents, such as [39, 41], and in a large amount of grey literature [4, 38, 60, 64].

Furthermore, software is also exposed at the expansion of its original intention well beyond the initial ideas, as it is evident for instance in the history of PowerPoint, well described by Brock [20].

Finally, software development has also then been exposed to the four fundamental approaches to handle multiplicity:

1. *single inspiration but multiple interpretations*: this reflects the idea of describing a system as a set of different and coherent views, as it was done, for instance, in the mid-90s with the methodologies developed around the newly defined UML[61], since UML allows to have multiple diagrams on the system being developed, such as class diagram, object diagram, interaction diagram, etc; moreover, UML supports some partial checking of the coherence existing among these diagrams;
2. *multiple narrations* proceeding in parallel: this appears to be quite present in agent-based systems, like the ones advocated since the beginning of the appearance of the theory of agents [26]; also nowadays approaches based on swarm intelligence and genetic algorithms resemble strongly this approach;
3. *single and universal description*: this is what happens when we try to build an omni-comprehensive system; as already mentioned, it is not a prerogative of only waterfall methods, but, for instance, it applies also to the mentioned early object oriented approaches, the ones occurring before the advent of agile methods;
4. collection of *non contradictory simple models*: this resembles strongly agile methods collecting requirements in the forms of simple users stories, as very well described by Kelly [44] and then developing incrementally the minimal amount of functionality needed to satisfy the stakeholders of such user stories.

7.3 Additional Lessons to Learn from the Reflections of Calvino on Multiplicity

During this discussion Calvino mentions the concept of the "hyper-romance," that is a romance that is a combination of multiple plots, also with multiple starting points or ending points. This structure could be an interesting reference point for what we have in software as frameworks for creating different applications, and could provide and interesting reference for them.

At the end of the discussion of multiplicity, Calvino mention that he would like to perceive the romance as a network of ideas, and even of contributions from different people, like an emerging entity. In software, agile methods have emphasized the concept of emerging architectures. Open source communities have created software based on various contributions of different people. Still, much more reflections could be done in this area, also taking advantages of less structured languages.

8 Discussion

At the end of this review of the work of Calvino we can say that definitely several recommendations, prescriptions, and guidelines that have been proposed to software writers were already present in the work of a literary critic as Calvino. Moreover, from his work we can identify several issues that cross his lectures and that could be profitably considered in our discipline.

The first, is about the description by under-specification and by vagueness; across his lectures Calvino goes over this concept many time. With respect to computer science, we know that in formal models non determinism allows sometimes simpler, lighter, and more understandable descriptions than fully deterministic structures, e.g., considering non deterministic finite state machines vs. their counterparts [32]. Also in machine learning it has been found that random gradient descent approaches can be more effective than deterministic approaches [57]. We already are fully aware that limiting the description facilitates variability, however we could move further, focusing on the "power of not saying;" such approach could appear counterintuitive, going against decades of claims that requirements need to be fully specified, or, at least, specified in an incremental way, but still appears quite promising.

The next point is about rapidity in writing. It is already well known that it is difficult to conclude software projects, that it is important to shorten the time to markets of products, that we need to reduce the effort to save resources. However, Calvino mentions something innovative: being fast is a valuable resource, in the sense that certain activities cannot be performed slowly. In other terms, if we cannot do them fast, we will never be able to do even with an infinite amount of time. Sometimes people call this inspiration, Plato wrote an entire dialogue on this (the Crito); it something that has to be captured while it is "flying." However, we have never considered this in software, and also this is counterintuitive. We have always considered time as an additive resource, the more we have, the more we can do.

Calvino also emphasises in multiple ways that generalizing is intrinsically a bogus process. We may aim at some

forms of abstractions, however, only detailed descriptions can capture the reality. Agile evangelists have warned about the "super generalizations" of the early era of object orientation, that typically lead to failures. However, Calvino goes further: like for speed in development, he claims that there are properties can can be captured only by focused and not general descriptions. Perhaps, in writing software we should put a similar warning.

Calvino then promotes the creations of visual models and of metaphors beyond what is typical of software engineering. On one side he describes the emergence of visual representation as a community and social process that is always ongoing, and it should be so for the health of the discipline. On the other, it promotes a discipline in itself of creating vivid images of the reality, not amorphous abstractions, as a way to gather a better understanding of what we need to model.

Finally, Calvino acknowledges that narrations are like live entities, they often never end and have a life on their own, well beyond the initial intentions of the writer(s), and even sometimes with a structure that is not predefined and it emerges as the writing progresses. This is also very true for software and we have evidences of such evolution, and even a specific area of software engineering named exactly software evolution. Still, the idea of planning software for its never ending completion or its evolution is still at its infancy, and this does not relate purely on how architecting and designing software systems, but also with reference to organization of the teams, licensing and revenue schemas, maintenance, and several other [35, 42, 53].

9 Conclusion

In this paper we have made two major points. The **first** one is that literature in general, and, specifically, the discipline of Poetics can provide inspiration for the advancement of software engineering, and we have argued this after an initial reference on Aristoteles, analysing the work of Calvino "Six Memos for the Next Millennium." The **second** one is that such work raises several points that have been partially covered (mostly later) by software engineering in general, and especially by the recent agile movement, and in part are still unexplored.

Needless to say, the present work is a collection of *reflections*, hence the title. It is the proposal for avenues for new investigations, and, as such, it is intended as the start of a deeper and more pervasive exploration of the implications and the contributions that may come from Poetics, and also Rhetoric, Aesthetic, etc. In this work we do not focus on such aspects because a full review of the influences that the discipline of poetic (may) have on software development is a very large work, definitely beyond the scope of this analysis. Rather, we focus on one specific work, the one by an Italian writer of the 20th century who appears to have a strong correlation, and perhaps somehow also influence, on the latest trends of software development.

Additional interesting and relevant input may also from other artistic disciplines, such as Visual Arts, Dancing, Acting, Music, and so on, that have already shown potentials for beneficial contributions to software [33, 55], and should be subject of a very careful scrutiny.

Acknowledgments

The authors thanks Innopolis University for generously supporting this research. The first author thanks CINI for partial support under contract AMINSEP. The third author thanks Francesco Martinelli for explaining the overarching role of this Calvino's work, namely *Six memos for the next millennium.*

References

[1] Aristotles. 335BC. Poetics. (335BC).
[2] John Backus. 1978. Can Programming Be Liberated from the Von Neumann Style?: A Functional Style and Its Algebra of Programs. *Commun. ACM* 21, 8 (1978), 613–641.
[3] Donald Bagert and Nancy Mead. 2001. Software Engineering as a Professional Discipline. *Computer Science Education* 11 (01 2001), 73–87. DOI: http://dx.doi.org/10.1076/csed.11.1.73.3841
[4] Ian Bagost. 2015. Programmers: Stop Calling Yourselves Engineers. (Nov 2015). https://www.theatlantic.com/technology/archive/2015/11/programmers-should-not-call-themselves-engineers/414271/ Visited on 20 August 2020.
[5] A. James Baroody, Jr. and David J. DeWitt. 1981. An Object-oriented Approach to Database System Implementation. *ACM Trans. Database Syst.* 6, 4 (1981), 576–601.
[6] V. Basili, L. Briand, and others. 1996. Understanding and predicting the process of software maintenance releases. In *Proc. 18th Int. Conf. on Software Engineering*. IEEE, 464–474.
[7] V. Basili, G. Caldiera, and H. Dieter Rombach. 1994. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley.
[8] Kent Beck. 1997. *Smalltalk Best Practice Patterns.* Prentice Hall.
[9] Kent Beck. 1999. Extreme Programming: A Discipline of Software Development. In *Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-7)*. Springer-Verlag, Berlin, Heidelberg, 1–. http://dl.acm.org/citation.cfm?id=318773.318778
[10] Kent Beck. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, Boston, MA, USA.
[11] Kent Beck. 2003. *Test-driven Development: By Example.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[12] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 1999. Manifesto for Agile Software Development. (1999). Online: http://www.agilemanifesto.org, on 8th October 2020.
[13] L. Benedicenti. 2017. Chapter 6 - Introducing Ubiquity in Noninvasive Measurement Systems for Agile Processes. In *Adaptive Mobile Computing*, Mauro Migliardi, Alessio Merlo, and Sherenaz Al-Haj Baddar (Eds.). Academic Press, Boston, 109 – 126.
[14] G. Beniamini, S. Gingichashvili, A. Orbach, and D. Feitelson. 2017. Meaningful identifier names: the case of single-letter variables. In *Proc.*

25th Int. Conf. on Program Comprehension (ICPC). IEEE, 45–54.
[15] B. Boehm and R. Turner. 2003. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, Boston, MA, USA.
[16] B. Boehm and R. Turner. 2003. Observations on balancing discipline and agility. In *Proc. of the Agile Development Conference*. IEEE, 32–39.
[17] B. Boehm and R. Turner. 2003. Using risk to balance agile and plan-driven methods. *Computer* 36, 6 (2003), 57–66.
[18] Grady Booch. 1982. Object-oriented Design. *Ada Lett.* I, 3 (1982), 64–76.
[19] G. Booch. 2018. The History of Software Engineering. *IEEE Software* 35, 5 (2018), 108–114.
[20] D. C. Brock. 2017. The improbable origins of Powerpoint. *IEEE Spectrum* 54, 11 (November 2017), 42–49. DOI: http://dx.doi.org/10.1109/MSPEC.2017.8093800
[21] Italo Calvino. 1988. *Lezioni americane. Sei proposte per il prossimo millennio*. Garzanti.
[22] Italo Calvino. 1988. *Six Memos for the Next Millennium (translated by P. Creagh)*. Harvard University Press.
[23] Lodovico Castelvetro. 1563. *Poetica d'Aristotele vulgarizzata, et sposta (The Poetics of Aristoteles, translated and explained)*. Laterza.
[24] Roderick Chapman, Neil White, and Jim Woodcock. 2017. What Can Agile Methods Bring to High-integrity Software Development? *Commun. ACM* 60, 10 (2017), 38–41.
[25] Raffaele Fabio Ciriello, Alexander Richter, and Gerhard Schwabe. 2017. When prototyping meets storytelling: practices and malpractices in innovating software firms. In *2017 IEEE/ACM 39th Int. Conf. on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, Buenos Aires, Argentina, 163–172.
[26] William D Clinger. 1981. *Foundations of Actor Semantics*. Ph.D. Dissertation. MIT, Cambridge, MA, USA.
[27] A. Cockburn. 2001. *Agile software development*. Pearson.
[28] M. Conway. 1968. How do committees invent. *Datamation* 14, 4 (1968), 28–31.
[29] James O. Coplien and Gertrud Bjørnvig. 2010. *Lean Architecture: For Agile Software Development*. Wiley, Chichester, UK.
[30] Stephan Diehl. 2007. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media.
[31] Edsger W. Dijkstra. 1968. Letters to the Editor: Go to Statement Considered Harmful. *Commun. ACM* 11, 3 (1968), 147–148.
[32] M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu (Eds.). 2004. . Lecture Notes in Computer Science, Vol. 3317. Springer, Kingston, Canada.
[33] I. Erofeeva, V. Ivanov, S. Masyagin, and G. Succi. 2020. Learning agility from dancers – experience and lesson learnt. In *Devops2020*. Springer.
[34] Matthias Feurer and Frank Hutter. 2019. Hyperparameter Optimization. In *Automated Machine Learning: Methods, Systems, Challenges*, Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). Springer, 3–33.
[35] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou. 2014. Variability in Software SystemsâĂŤA Systematic Literature Review. *IEEE Transactions on Software Engineering* 40, 3 (March 2014), 282–306. DOI: http://dx.doi.org/10.1109/TSE.2013.56
[36] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[37] B. Hungerford, A. Hevner, and R. Collins. 2004. Reviewing software diagrams: A cognitive study. *IEEE Transactions on Software Engineering* 30, 2 (2004), 82–96.
[38] Mirona Iliescu. 2012. Software development never ends, unless you are doing it all wrong. (Dec 2012). https://metabroadcast.com/blog/software-development-never-ends-unless-you-re-doing-it-all-wrong/ Visited on 14th August 2020.
[39] Andrea Janes and Giancarlo Succi. 2014. *Lean Software Development in Action*. Springer, Heidelberg, Germany. DOI: http://dx.doi.org/10.1007/978-3-642-00503-9
[40] Andrea A. Janes and Giancarlo Succi. 2012. The Dark Side of Agile Software Development. In *Proce. ACM Int. Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2012)*. ACM, 215–228.
[41] Ron Jeffries. 2015. *The Nature of Software Development: Keep It Simple, Make It Valuable, Build It Piece by Piece* (1st ed.). Pragmatic Bookshelf.
[42] Haruhiko Kaiya, Ryohei Sato, Atsuo Hazeyama, Shinpei Ogata, Takao Okubo, Takafumi Tanaka, Nobukazu Yoshioka, and Hironori Washizaki. 2017. Preliminary Systematic Literature Review of Software and Systems Traceability. *Procedia Computer Science* 112 (2017), 1141âĂŞ1150.
[43] Alan C. Kay. 1993. The Early History of Smalltalk. *SIGPLAN Not.* 28, 3 (1993), 69–95.
[44] A. Kelly. 2017. *A Little Book of Requirements & User Stories*. Software Strategy Limited.
[45] Rilla Khaled, Pippin Barr, James Noble, and Robert Biddle. 2004. System metaphor in Extreme Programming: A semiotic approach. In *Proc. 7th Int. Workshop Organisational Semiotics*. 1–24.
[46] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-oriented programming. In *ECOOP'97 — Object-Oriented Programming*, Mehmet Akşit and Satoshi Matsuoka (Eds.). Springer, Berlin, Heidelberg, 220–242.
[47] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2012. Technical debt: From metaphor to theory and practice. *IEEE Software* 29, 6 (2012), 18–21.
[48] Alexander Laufer, Terry Little, Jeffrey Russell, and Bruce Maas. 2018. *The Agility Practice: Be Responsive and Action Oriented*. Springer International Publishing, Cham, 31–54. DOI: http://dx.doi.org/10.1007/978-3-319-66724-9_3
[49] Peter Lloyd. 2000. Storytelling and the development of discourse in the engineering design process. *Design Studies* 21, 4 (2000), 357–373.
[50] Gabriele Lolli. 2011. *Discorso sulla matematica: una rilettura delle Lezioni americane di Italo Calvino*. Bollati Boringhieri. https://books.google.ru/books?id=KtJGAQAAIAAJ
[51] Gabriele Lolli. 2013. Mathematics according to Italo Calvino. In *Imagine Math 2: Between Culture and Mathematics*, Michele Emmer (Ed.). Springer, Milan, 49–56.
[52] L Luque, E d S Veriscimo, G d C Pereira, and LVL Filgueiras. 2014. Can we work together? on the inclusion of blind people in uml model-based tasks. In *Inclusive Designing Joining Usability, Accessibility, and Inclusion*. Springer, 223–233.
[53] Ruchika Malhotra and Anuradha Chug. 2016. Software Maintainability: Systematic Literature Review and Current Trends. *International Journal of Software Engineering and Knowledge Engineering* 26 (10 2016), 1221–1253. DOI: http://dx.doi.org/10.1142/S0218194016500431
[54] Robert C Martin. 1996. The dependency inversion principle. *C++ Report* 8, 6 (1996), 61–66.
[55] S. Masyagin, M. Nurgalieva, and G. Succi. 2019. Kent Beck or Pablo Picasso? – Speculations of the relationships between artists in software and painting. In *TOOLS50+1 2019*.
[56] Karin Müller. 2012. How to make UML diagrams accessible for blind students. In *Proc. Int. Conf. on Computers for Handicapped Persons (LNCS)*, Vol. 7382. Springer, 186–190.
[57] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. 2009. Robust Stochastic Approximation Approach to Stochastic Programming. *SIAM Journal on Optimization* 19, 4 (2009), 1574–1609.
[58] D. L. Parnas. 1972. On the Criteria to Be Used in Decomposing Systems into Modules. *Commun. ACM* 15, 12 (1972), 1053–1058.
[59] Klaus Pohl and Andreas Metzger. 2006. Variability management in software product line engineering. In *Proc. 28th Int. Conf. on Software Engineering*. IEEE/ACM, 1049–1050.
[60] Daniel Riedel. 2015. Why Software Development Is Never Over. (Sept 2015). https://www.corpmagazine.com/industry/technology/software-

development-never/ Visited on 20 August 2020.
[61] J. Rumbaugh, I. Jacobson, and G. Booch. 1998. *The Unified Modeling Language Reference Manual.* Addison Wesley, Reading, MA, USA.
[62] E. Schonberg, N. Mitchell, and G. Sevitsky. 2010. Four Trends Leading to Java Runtime Bloat. *IEEE Software* 27, 01 (2010), 56–63.
[63] M. Shaw. 1990. Prospects for an Engineering Discipline of Software. *IEEE Software* 6, 7 (1990), 15–24.
[64] Shamoon Siddiqui. 2014. Why developers never finish their projects. (Mar 2014). https://medium.com/things-developers-care-about/why-developers-never-finish-their-projects-bf39d3424114/ Visited on 14th August 2020.
[65] J. Spolsky. 2005. The Project Aardvark Spec. Wite Paper, Fog Creek, published on Joel on Software's blog. (2005).
[66] B. Srinivasan. 2012. A for Agile, A for Aristotle. https://www.agileconnection.com/article/agile-aristotle. (2012).
[67] Jean Starobinski. 2001. *La Relation critique.* Gallimard.
[68] Sven Theobald and Philipp Diebold. 2017. Beneficial and Harmful Agile Practices for Product Quality. In *Procs. 18th Int. Conf. Product-Focused Software Process Improvement (PROFES)*, M. Felderer and others (Eds.). Springer, 586–593.
[69] David Turner. 1986. An Overview of Miranda. *SIGPLAN Not.* 21, 12 (1986), 158–166.
[70] A. VanLamsweerde. 2001. Goal-oriented requirements engineering: A guided tour. In *Procs. 5th Int. Symposium on Requirements Engineering*. IEEE, 249–262.
[71] Joseph White. 2013. *St. Therese of Lisieux: Meditations with the Little Flower.* Our Sunday Visitor. https://www.xarg.org/ref/a/1612785913/
[72] J. Zhi, V. Garousi-Yusifoğlu, B. Sun, G. Garousi, S. Shahnewaz, and G. Ruhe. 2015. Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software* 99 (2015), 175–198.