Pontifícia Universidade Católica de São Paulo
PUC-SP

Francisco Supino Marcondes

Uma Abordagem Bergsonista para o Estudo da Programação

Mestrado em
Tecnologias da Inteligência e Design Digital

São Paulo
2015

Pontifícia Universidade Católica de São Paulo
PUC-SP

Francisco Supino Marcondes

Uma Abordagem Bergsonista para o Estudo da Programação

Mestrado em
Tecnologias da Inteligência e Design Digital

Dissertação apresentada à Banca Examinadora da Pontifícia Universidade Católica de São Paulo, como exigência parcial para obtenção do título de MESTRE em Tecnologias da Inteligência e Design Digital, na área de concentração de Processos Cognitivos e Ambientes Digitais, na linha de pesquisa de modelagem de Sistemas de Software, sob a orientação do Prof. Dr. Ítalo Santiago Vega.

São Paulo
2015

Banca Examinadora:

_____

_____

_____

Ao mestre.

Agradecimentos

O que seria
da vida sem a ajuda do D'us Messias?
Assombrosos foram os milagres que só eu vi!

O que seria
de mim sem minha esposa? Existem coisas
em que não se deve pensar, e esta é uma delas.

O que seria
do quotidiano sem meus pais e filhos?
Penso que o dia a dia perderia toda a emoção.

Acho que sem D'us,
minha esposa e família todo o esforço
empreendido neste estudo seria em vão. Muito Obrigado!

Agradeço em especial ao prof. Luiz Alberto Vieira Dias que me permitiu ingressar nessa linha de questionamento. Ao prof. Eduardo Cardoso Braga por ter sido apoio determinante na exploração inicial. Ao prof. Ítalo Santiago Vega que conduziu com rigor e gentileza o desenvolvimento do tema. E à Edna Conti, sem a qual, certamente esta dissertação não existiria.

"lálàia láia láia lálàia láia láia lálàia láia
láia lá lá lá lá lá láiá-á."
— Jair Rodrigues

Resumo

Este estudo discute a dificuldade que se tem em abordar a programação de maneira científica e sugere uma forma de fazê-lo. Talvez a maior dificuldade que se tem ao abordar a programação seja o fato desta manifestar-se como um objeto metafísico onde o método científico não pode ser aplicado. Dentre as possibilidades para abordar objetos metafísicos escolheu-se a proposta metodológica de Henri Bergson por se propor a ser tão preciso na investigação metafísica quanto o método científico é na investigação científica. O bergsonismo se fundamenta na suposição de durações por meio de um esforço de intuição. Intuição é uma forma de cognição sofisticada que se manifesta apenas quando há um alto grau de especialização frente à um objeto. A precisão bergsonista consiste na possibilidade de, dado um contexto, diferentes pessoas suporem uma mesma duração, onde, este conjunto de suposições semelhantes caracteriza inter-subjetividade, ou seja, a objetividade científica. Assim, com base no bergsonismo foi derivado um método para o estudo da programação o qual foi aplicado em uma prova de conceito. Embora na prova de conceito o bergsonismo tenha se mostrado bem sucedido no estudo da programação, este é ainda um resultado preliminar. Portanto, ainda são necessários novos estudos para averiguar a real contribuição do bergsonismo no estudo da programação.

Palavras-chave: Ciência da Programação. Bergsonismo. Metodologia Científica.

Abstract

This paper discusses the difficulties to scientifically study computer programming and suggests a way to perform it. A difficulty that must to be dealt is the fact that programming is a metaphysical object in which scientific method is not aplicable. Among several possibilities capable to study metaphysical objects, Bergson's methodological approach was choosen since it expects to be as rigorous in metaphysical research as scientific method in scientific research. Bergsonism explores metaphysical objects supposing durations by an intuition effort. Intuition is a sophisticate cognition that emerges only there is a high skill level with an object. Bergsonism precision consists in different people reach, through intuition, similar durations in a given context. Those similar durations yields intersubjectivity, i.e., scientific objectivity. Based on bergsonism an research approach was derived and applied on a proof of concept. Even bergsonism appearing to be successful to programming research programming this is yet a preliminary result. Some more research is need to evaluate the adequacy of bergsonism to programming research.

Keywords: Programming Science. Bergsonism. Scientific Methodology.

Prefácio

A proposta inicial deste estudo era conceber uma arquitetura de referência a qual apoiasse o desenvolvimento de sistemas voltados para a modelagem e simulação do clima espacial. A justificativa para a realização deste estudo pode ser resumida na realidade de que embora os sistemas de software voltados a tal atividade sejam complexos por natureza a ponto de seu número de estados passar facilmente da casa dos milhões, são produzidos em geral, por físicos espaciais, *i.e.*, pessoas sem a formação necessária para a produção apropriada de sistemas de software de tal complexidade. Dentre os diversos problemas conhecidos (como baixa escalabilidade, difícil manutenção, pouca robustez, *et cetera*) talvez o pior problema possível seja a baixa confiança trazida pelos dados resultantes das simulações quando construídos sobre uma arquitetura favela. Em especial este problema tem um impacto considerável para esta área de estudo podendo inclusive atrasar o seu desenvolvimento pois trata-se de uma área onde com grande velocidade hipóteses são propostas e descartadas tendo o computador como uma importante (senão fundamental) ferramenta de trabalho ajudando a comprovar ou refutar hipóteses por meio de simulações programadas pelos cientistas.

A solução proposta era criar uma arquitetura de referência que pudesse ser facilmente especializada dando origem a uma linha de produtos (Software Product Line - SPL) à la e buscando aumentar ainda mais a confiabilidade de tal modelo foi proposto que deveria incorporar métodos formais. O cientista poderia então partir sempre desta arquitetura inicial com a intenção de evitar a propagação de erros entre diferentes versões subsequentes de arquitetura favela. Outrossim percebeu-se que as linguagens de programação de propósito geral não apoiariam o físico espacial para a especialização apropriada da arquitetura de referência, optou-se então pela criação de uma linguagem de domínio específico a qual, por um lado ocultasse o máximo possível complexidades tecnológicas e por outro que facilitasse a fluidez de leitura/escrita do código. Naturalmente o sucesso deste objetivo tem correlação direta com o grau de especificidade da arquitetura proposta, *i.e.*, se muitos elementos fossem previamente definidos a arquitetura engessaria a SPL restringindo os software que poderiam ou não ser derivados dela, enquanto que, por outro lado, se deixada de forma muito genérica qualquer sistema de

software poderia ser derivado dela sendo que nenhuma situação era desejável sendo que o meio termo seria definido em comum acordo com um pesquisador da área. Pode-se argumentar que uma alternativa mais interessante à de uma SPL seria a construção de um framework, no entanto, ele recai no problema determinar o grau de especificidade, e embora possa-se chegar num acordo sobre o grau de especificidade desejada, caso haja uma mudança nessa necessidade o framework terá de ser modificado de qualquer forma. Como trata-se de uma área em que o risco disso acontecer é grande e a atualização de versões de arquiteturas de referência tendem a ser menos custosas, optou-se pelo uso de SPL.

Ao iniciar os estudos para compor a revisão bibliográfica, o autor optou primeiro por detalhar cada um dos termos presentes do título o que, como esperado, levou o autor à alguns questionamentos os quais direcionaram o restante do estudo. O primeiro questionamento neste sentido foi a tentativa de definir o termo arquitetura presente no título da proposta inicial, grosso modo, o resultado do levantamento bibliográfico foi o mesmo descrito por Fowler onde:

> A indústria de software se delicia em pegar palavras e estendê-las em uma miríade de significados sutilmente contraditórios. Uma das maiores sofredoras é a arquitetura. Vejo arquitetura como uma daquelas palavras que soam impressionantes, usadas principalmente para indicar que estamos falando de algo importante. (...) Arquitetura é um termo que muitas pessoas com pouca concord^ancia entre si tentam definir. Fowler (2002)

Esta afirmação remove do estudo a possibilidade de propor uma arquitetura pois considerando que não existe um consenso, cada pesquisador, partindo de uma definição diferente, tende a rapidamente degenerar a arquitetura numa favela e isso seria um impedimento ao objetivo inicial. Embora não exista um consenso sobre o que é ou o que deve conter uma arquitetura, existe um consenso de que ela realiza, em termos de modelo, uma lista de requisitos. Decidiu-se então que a opção mais razoável que ficasse próxima ao objetivo inicial seria a definição de um conjunto de requisitos que pudessem ser usados como referência para a criação de arquiteturas de software. Ao fazer o levantamento bibliográfico percebeu-se que novamente não existe um consenso sobre o que seja um requisito sendo que inclusive numa mesma obra é apresentada mais de uma definição diferente. Outro ponto que requer atenção é que na idéia de requisitos de referência implica que duas empresas tem os mesmos requisitos e sendo verdade significa que elas tem os mesmos processos de negócio (no nível micro) e por inferência lógica isso é verdade se e somente se forem a mesma empresa e em conversas com profissionais da área em percebe-se que tal afirmação é verdadeira e considerando

que a arquitetura realiza requisitos, percebeu-se que não é possível igualmente obter sucesso na criação de uma arquitetura de referência com o objetivo proposto. Em outras palavras tal levantamento remove do título o termo referência; um possível fator da dificuldade de se obter consenso sobre a definição de requisitos está no fato dela estar atrelada ao conceito de software, i.e., requisitos de software. O problema deste atrelamento é que também não há um consenso sobre o que seja software, sendo inclusive uma discussão em congressos de filosofia da computação ainda em 2009.

Por fim, percebeu-se que o uso de métodos formais não traz maior corretude ao modelo apresentando muitas vezes os mesmos problemas que os métodos não formais sendo apenas mais complicados. Tal realidade pode ser entendida considerando o teorema da incompletude de Gödel. Este mesmo teorema remove da DSL o objetivo de representar com completude um pensamento, outrossim, a facilidade e fluidez de uma linguagem é altamente subjetivo sendo que tal ferramenta dificilmente ajudaria a atingir os objetivos originais do estudo. Neste contexto percebe-se que o autor ficou numa posição bastante frágil sobre como abordar o problema até o surgimento do Software Engineering Method and Theory (SEMAT) onde a confusão do autor pôde ser confirmada por pesquisadores de bastante renome. Decidiu-se portanto antes de prosseguir com o estudo que era necessário um levantamento aprofundado da bibliografia visando entender com maior profundidade o problema enfrentado pela área de Engenharia de-Software para aí sim ser possível apresentar uma hipótese apropriada redirecionando o estudo.

Como resultado percebeu-se que possivelmente o estudo sobre micro processos seria mais adequado uma vez que, segundo Parnas (1986), é impossível obter um macro processo que seja racional para o desenvolvimento de software. Do ponto de vista de micro processo, percebeu-se que grande parte dos problemas resolvidos atualmente com software não tem relação com a matemática mas que em algum momento ele precisa ser formalizado considerando que o código fonte não passa de uma linguagem formal. O problema disso é que em geral as técnicas de engenharia de requisitos (o que quer que seja um requisito) acabam sofrendo influência da matemática onde muitos aspectos da aplicação são perdidos já na análise; para um exemplo bastante comum, considere a análise de um processo de negócio onde o analista tenta especificar de maneira boleana uma decisão que na realidade é fuzzy, ou seja, acaba-se reduzindo o problema à algo que ele não é e portanto nada mais natural do que as atuais (2010) taxas de insucesso nos projetos de software.

Embora o autor não tenha encontrado nenhuma afirmação direta sobre isso, percebe-se em especial na análise de domínio (uma das fases da Engenharia de Requisitos) o predomínio de métodos heurísticos o que talvez seja um indicativo desta realidade, como

resultado disso, tal atividade não é feita de maneira rigorosa ou sistemática, onde o sucesso do modelo de domínio depende mais da habilidade individual do analista do que fruto de uma abordagem sistemática. Decidiu-se que uma investigação interessante poderia ser justamente a de usar um sistema de classificação rigoroso embora não matemático para apoiar o a tal atividade e durante tal busca percebeu-se que tal investigação poderia ter como base a Filosofia da Mente como proposta por Henri Bergson.

Sumário

Lista de Figuras

Capítulo 1

Introdução

A forma de conhecimento proporcionado pela ciência, tem impulsionado o desenvolvimento tecnológico da civilização ocidental ao longo dos últimos séculos. Por isso, desde os primórdios da computação, houve um esforço deliberado e intencional no sentido de fazer com que a programação passasse a ser reconhecida como ciência [Knuth, 1974]. Possivelmente, a origem deste esforço foi a busca pelo estabelecimento de uma engenharia (ciência aplicada) de software [Shaw, 1990], a qual, por sua vez, buscava atacar o problema que ficou conhecido como "crise do software". A crise do software consiste na dificuldade de saber *a priori* se um artefato antederá as espectativas que se tem sobre ele [Naur & Randell, 1969].

Em outras palavras, para atacar a crise do software, no final da década de 1960 surgiu a necessidade do desenvolvimento da engenharia de software [Naur & Randell, 1969]. E, da necessidade do desenvolvimento da engenharia de software, no início da década de 1970 surgiu a necessidade de desenvolver a Ciência da Programação [Shaw, 1990]. Cabe dizer que o esforço empregado até o momento não foi bem sucedido [Johnson *et al.*, 2012] de forma que, ainda hoje, este seja tema para discussão [Jacobson & Spence, 2009, Jacobson & Meyer, 2009].

A Ciência da Programação busca explicar como se dá a criação do conjunto funções de transição parcial (δ), que fazem uma Máquina de Turing, ao computar, apresentar um comportamento de interesse [Sipser, 2012]. Isto é, dada uma Máquina de Turing, entende-se como *programação* as atividades voltadas para a definição de um conjunto de funções de transição parcial, visando que a máquina realize um comportamento de interesse. E entende-se como *computação*, o comportamento apresentado pela máquina ao realizar um conjunto de funções de transição parcial, definidos durante a programação.

Decorre daí, que a *ciência da computação* estuda os possíveis comportamentos da máquina dado um programa, enquanto a *Ciência da Programação* estuda como criar

um programa que faça a máquina apresentar um comportamento de interesse. Neste estudo, *o objeto da pesquisa é a Ciência da Programação.*

1.1 Método Científico em Programação

O conhecimento científico é fundamentado no método científico e este é aplicado sobre fenômenos. Entende-se fenômeno como uma ocorrência observável, mas, considerando que a observação não é necessariamente visual, um outro entendimento equivalente seria de que o fenômeno científico é uma ocorrência mensurável [Bergson, 2006]. O método científico consiste em um processo sistemático de colocar e testar hipóteses com base em fenômenos [Popper, 2004]. Tanto melhor será a hipótese quanto maior sua precisão na previsão e explicação de fenômenos.

Em ciência, a precisão de uma hipótese é determinada pelo que se conhece por "objetividade científica". A objetividade científica, por sua vez, é determinada de maneira intersubjetiva. Ou seja, são as pessoas de uma mesma área de conhecimento e época que verificam a hipótese e a tomam como precisa [Kuhn, 2005]. Dada uma hipótese que foi corroborada, ela pode sofrer um processo de refinamento sucessivo, onde suas falhas vão sendo descobertas e corrigidas.

Portanto, é sugerido que a Ciência da Programação seja uma forma de conhecimento fundamentada na aplicação sistemática do método científico, sobre os fenômenos da programação.

A dificuldade desta concepção reside no fato de que, embora o resultado da programação (isto é, o programa) seja mensurável [DeMarco, 1986], a programação propriamente dita não o é. A programação é um processo sócio-cognitivo [Dijkstra, *s.d.*, Weinberg, 1998] que, em princípio, não pode ser mensurado. Mesmo que se obtenha uma forma de mensuração, dificilmente um experimento sócio-cognitivo em particular poderá ser reproduzido ou verificado. Isto significa que o método científico tradicional *cf.* [Popper, 2004] não é aplicável no estudo de processos sócio-cognitivos.

Esta é a base de algumas das críticas feitas sobre o uso do termo ciência nas "ciências sociais" e "ciências cognitivas". Esta mesma crítica pode ser feita à idéia de ciência no contexto de "Ciência da Programação". A validade desta crítica, no contexto da programação, é percebida ao considerar que, mesmo após mais de quarenta anos de esforços dedicados neste sentido, ainda não existem boas hipóteses que expliquem ou prevejam os fenômenos de programação [Johnson *et al.*, 2012].

1.1.1 Alternativas ao Método Científico

Uma vez que a compreensão sobre algo é intersubjetivo, com o passar o tempo uma mesma ideia assume conotações diferentes [Kuhn, 2005]. Houve uma época, principalmente durante o século XX, em que um conhecimento só poderia ser considerado científico, se seguisse o método científico tradicional *cf.* [Popper, 2004]. Em um período anterior a este, a ideia de ciência se confundia com a de arte [Knuth, 1974]. E isto se repetiu inúmeras vezes no decorrer do tempo.

No século XXI, a ciência vem assumindo uma ideia voltada ao conhecimento obtido através da aplicação sistemática de algum método, com algum tipo de embasamento teórico e uma argumentação razoavelmente rigorosa [Feyerabend, 2010]. Por exemplo, nas ciências sociais existem diferentes métodos que podem ser aplicados como, por exemplo, experimento, *survey*, análise de arquivos, pesquisa histórica e estudos de caso [Yin, 2015]. Nas ciências cognitivas são utilizados métodos diversos para a exploração da mente, como os usados na psicologia, neurociência, antropologia, inteligência artificial, filosofia, *et cetera* [Posner, 1993].

Assim, para cada ciência em que o método científico não se aplica, da comunidade de especialistas daquela área do conhecimento emerge uma forma reconhecidamente apropriada, com a qual se consegue abordar o objeto de estudos [Shaw, 1990]. Isto sugere que a realização de um estudo científico sobre a programação requer a definição de um método que permita ao investigador abordá-lo de maneira adequada. Neste sentido, *o problema tratado nesta pesquisa é o de propor uma abordagem que permita estudar a programação de forma científica*, ou ao menos, o mais próximo possível à isso.

1.2 Métodos para a Investigação da Programação

Uma vez que a Ciência da Programação visa produzir conhecimentos científicos relacionados à atividade de programação, é preciso estabelecer um método que possa ser sistematicamente aplicado em seu estudo. Duas características desejáveis em um método proposto para estudar a programação, é que ele produza hipóteses que possam ser refinadas e que os resultados produzidos sejam objetivos. Com o refinamento das hipóteses, se espera alcançar um conhecimento cada vez mais preciso na explicação e previsão dos fenômenos da programação. Com a objetividade, se espera conseguir verificar e validar, intersubjetivamente, os resultados obtidos durante a investigação.

Tendo por base estes critérios, convém explorar algumas possibilidades de métodos que propiciem estudar a programação e tentar supor o motivo deles não terem se con-

solidado como uma abordagem adequada para o estudo da programação. Independente da concepção de ciência e fenômeno que se estiver usando, dado que a programação é um "fenômeno" sócio-cognitivo, sugere-se que a "ciência" da programação possa ser investigada, no sentido de avaliar se alguma de suas alterativas metodológicas convém ao estudo da programação.

1.2.1 Abordagens Quantitativa e Qualitativa

Se a programação for um processo sócio-cognitivo, uma alternativa razoável de investigação seria por meio das ciências cognitivas. Dentre as possibilidades metodológicas desta ciência, é também razoável investigar a possibilidade da aplicação dos métodos utilizados pela psicologia. Grosso modo, a psicologia utiliza as abordagens quantitativas e qualitativas para a realização deste tipo de estudo [Silva, 2010].

A abordagem quantitativa consiste na coleta e avaliação de métricas observadas no comportamento da pessoa [Silva, 2010]. Esta abordagem foi utilizada no estudo da programação por Humphrey e consistia na coleta de, por exemplo, métricas de tempo e defeito [Humphrey, 2005]. Com base no acompanhamento sistemático dessas métricas e fazendo uso de fórmulas estatísticas, foi possível criar um método que tanto é capaz de prever eventos futuros, como refinar o processo de desenvolvimento. Embora tenha sido possível obter um resultado bastante eficiente [Humphrey, 2005], os resultados obtidos por um indivíduo não é aplicável aos outros. Em outras palavras, isto significa que o resultado não é objetivo. Portanto, não contribui para o entendimento científico sobre como ocorre a programação.

A abordagem qualitativa consiste na formação de um entendimento sobre um particular processo social ou cognitivo e consiste na realização ou aplicação de entrevistas, questionários, *et cetera* [Silva, 2010]. Esta abordagem foi utilizada no estudo da programação por Weinberg, que foi capaz de mapear e descrever diferentes cenários de programação, dos quais conseguiu extrair informações importantes no entendimento sobre a programação [Weinberg, 1998]. Embora esta iniciativa também possa ser considerada bem sucedida, os resultados obtidos são pontuais. Além disso, como a investigação depende da capacidade pessoal do investigador, o resultado não pode ser verificado ou refinado. Assim, este método tampouco contribui para o entendimento científico sobre como ocorre a programação.

1.2.2 Abordagem Empírica

A percepção da possível complementaridade entre as abordagens quantitativa e qualitativa é inevitável e sua união leva ao que se conhece por abordagem empírica, de

forma que: “o quantitativo se ocupa de ordens, grandezas e suas relações e o qualitativo formula um quadro de interpretações para medidas, ou a compreensão para o que não é quantificável” *cf.* [Silva, 2010]. Esta abordagem é a base do que se conhece como “estudos empíricos em engenharia de software” [Runenson & Höst, 2009].

A abordagem empírica, no estudo da programação, tem como base a abdução. Isso significa que seu método consiste em primeiro coletar os dados e, com base neles, imaginar qual a melhor explicação para os dados [Peirce, 2012], formulando assim uma hipótese. A hipótese é então generalizada por meio de um pensamento indutivo. Isto significa assumir que uma conclusão local tenha caráter universal [Popper, 2004]. A fragilidade do método empírico se deve, justamente, aos pensamentos abdutivos e indutivos. Tando a abdução como a indução, levam a conclusões pouco rigorosas e subjetivas, de forma que não podem ser refinadas nem verificadas. Cabe ressaltar que tal afirmação é adequada no âmbito da ciência e, portanto, não há objeção de que sejam usadas em sistemas formais *cf.* [Popper, 2004].

O uso do método empírico, para o estudo da programação, recai nos problemas apontados nos métodos quantitativo e qualitativo. Significa que é muito pouco provável que um conjunto de métricas e cenários válidos para um conjunto de pessoas, seja aplicável para qualquer outro. Ou ainda, para o mesmo grupo em um tempo e local diferente. Por fim, isso significa que o método empírico não contribui de maneira cabal para o entendimento cientifico da programação.

Em outros termos, a crítica a ser feita sobre o empirismo, como utilizado na psicologia para o estudo da programação, é que ela parte do fenômeno e tenta atingir a consciência, enquanto que o ideal seria o movimento oposto. Significa que o método usado na psicologia *cf.* [Silva, 2010] parece não se adequar apropriadamente ao estudo da programação. Por isso a investigação do processo de programação se evade do domínio da psicologia. Em ciência, este movimento oposto ao empírico caracteriza uma abordagem teórica. Cabe então investigar, ainda no escopo das ciências cognitivas, uma alternativa teórica que possa ser utilizada no estudo da programação.

1.2.3 Abordagem Teórica

Uma alternativa ao método empírico é a abordagem teórica. Entende-se teoria como uma explicação sobre o objeto, obtida através de reflexão racional [Suppe, 1998]. Embora a abordagem teórica pressuponha uma inferência inversa à apresentada pela empírica, ambas abordagens estão relacionadas. Esta afirmação se deve ao fato da abordagem teórica requerer a existência de evidências empíricas que a confirmem, enquanto a empírica necessita da formulação de uma hipótese que explique os dados [Suppe, 1998].

Isto é, a abordagem teórica parte de uma hipótese ou teoria e busca evidências que a confirmem; a abordagem empírica parte das evidências e apresenta uma hipótese que a explique.

O método científico tradicional *cf.* [Popper, 2004] segue a linha teórica e seu ponto de partida é a observação. No entanto, diferente do que ocorre na abordagem empírica, a observação do objeto não é pontual ou contextualizada, mas contínua. O desenvolvimento deste tipo de conhecimento é o descrito por Dreyfus, em seu modelo do desenvolvimento de habilidades [Dreyfus, 1998].

Segundo o modelo de Dreyfus, caso uma pessoa nunca tenha tido contato com um objeto se diz que ela é novata. Neste nível a pessoa interage com o objeto buscando reproduzir a forma com que outras pessoas interagem com ele. Ainda segundo este modelo, pela consecução das interações entre pessoa e objeto, a pessoa aumenta progressivamente sua habilidade junto ao objeto. Para atingir o nível mais avançado deste modelo, a pessoa deve ter sido capaz de apreender o objeto, de forma que não haja mais necessidade de pensamento consciente durante a interação. Em outras palavras, no nível inicial a interação da pessoa com o objeto se dá por imitação, nos níveis intermediários por inferência e no nível máximo por intuição [Dreyfus, 1998].

É neste sentido que o método empírico difere do teórico. A hipótese do método empírico, por derivar logicamente de um conjunto de dados, costuma ter origem em um dos níveis intermediários do modelo de Dreyfus. Por outro lado, a hipótese do método teórico, por ser resultado de intuição [Popper, 2004], costuma ter origem no nível mais sofisticado do modelo de Dreyfus. Isto não significa que um método seja cientificamente superior ao outro, mas que são métodos diferentes, com abordagens distintas. Uma vez que o método empírico não parece ser suficientemente adequado, no estudo da programação, convém explorar a possibilidade de uma abordagem teórica.

Cabe considerar que, como apresentado, o método teórico é o método científico. Este, por sua vez, também não se mostrou suficiente no estudo da programação. No entanto, a dificuldade do método científico não está no método em si, mas na mensuração dos fenômenos de programação. Quando o método científico é empregado em "fenômenos" qualitativos, o estudo deixa o âmbito da ciência e se insere no âmbito da filosofia [Deleuze, 1999].

Isto sugere que convém investigar, ainda no âmbito das ciências cognitivas, a alternativa metodológica usada pela filosofia da mente no estudo de seus objetos. Contudo, convém considerar também a afirmação de Bergson, de que os sistemas filosóficos existentes - pelo menos até o século XX - não são tão rigorosos e precisos quanto a ciência, ao conduzirem a investigação de seus objetos [Bergson, 2006].

1.2.4 Abordagem Conceitual

A crítica de Bergson aos sistemas filosóficos tem como base sua falta de precisão, onde uma mesma teoria pode ser aplicada tanto no contexto a que se propõe, como em qualquer outro [Bergson, 2006]. Na visão de Bergson, precisão em filosofia consiste em propor uma teoria que se aplique apenas a uma única coisa, tal qual ocorre na ciência [Bergson, 2006]. As leis de Maxwell, por exemplo, se aplicam única e exclusivamente ao eletro-magnetismo e nada mais. Assim também deveriam ser as teorias filosóficas.

A investigação de Bergson o faz concluir que o motivo disso está no raciocínio orientado por símbolos [Bergson, 2006]. Para Bergson, um símbolo é uma unidade discreta de conhecimento e por isso pode ser caracterizado como um signo peirceano. O pensamento conceitual é o resultado da manifestação do signo como um conceito. Um signo peirceano é um constructo mental formado por uma representação e uma interpretação sobre um objeto [Peirce, 2012]. Embora um mesmo objeto possa ter diferentes signos, cada um deles será uma abstração, uma compreensão em perspectiva do objeto. O raciocínio orientado por signos se dá por meio de semiose. Na semiose, de maneira sucessiva, se toma a interpretação de um signo como objeto e sobre ele se cria um novo signo [Peirce, 2012].

No estudo da programação, esta abordagem foi a escolhida pelo SEMAT, em sua iniciativa para conceber uma "teoria geral da engenharia de software". Sem rigor, se pode dizer que o SEMAT busca criar um *kernel* com os conceitos essenciais da engenharia de software e, também, uma sintaxe com a qual espera derivar os outros constructos [Jacobson *et al.*, 2013]. Para atingir este objetivo, o SEMAT vem envidando esforços no levantamento e catalogação das diversas práticas presentes na engenharia de software. Por meio deste catálogo, se pretende consolidar as práticas e assim formar o *kernel* [Jacobson *et al.*, 2013]. Esta forma de investigação caracteriza uma abordagem conceitual (ou simbólica).

A crítica, a ser feita ao SEMAT, é que esta escolha metodológica poderá lhe impedir de resolver muitos dos problemas aos quais se propôs a tratar. Considere, por exemplo, o problema em saber se dois métodos de programação são distintos ou equivalentes. A dificuldade de conseguir isso por meio da revisão conceitual de ambos é conhecida. O resultado é geralmente inconclusivo, uma vez que se se assemelham em alguns pontos e diferem em outros. A ausência de publicações sobre este tema serve como evidência desta dificuldade. Por outro lado, o problema seria facilmente resolvido se comparassem as vivências de programação desencadeadas por um método e por outro. O mesmo problema ocorre na filosofia. E é justamente essa a base da crítica de Bergson sobre eles, ao afirmar que lhes falta precisão [Bergson, 2006].

Este problema pode ser atribuído à semiose, pelo que Bergson chamou de "afastamento do real". Se diz que houve uma semiose, quando uma interpretação se torna o objeto de um outro signo. Assim, o processo de semiose produz um signo, que é a interpretação de uma interpretação de um objeto. Conforme este processo se repete, cada vez mais, a interpretação se distancia do objeto até que, em algum momento, o objeto real acaba sendo preterido. Este é um problema que Bergson denuncia nos sistemas filosóficos que se ocupam em criar interpretações sobre interpretações (ou signos sobre signos), até que se obtenha uma conclusão logicamente rigorosa, talvez com um excesso de escolhas arbitrárias de interpretações, mas que não tem qualquer sentido no contexto do real [Bergson, 2006].

Dito isso, Bergson sugere uma abordagem que se propõe em tratar este problema [Bergson, 2006]: o bergsonismo. Isso sugere que o bergsonismo deva, ao menos, ser considerado como possibilidade metodológica para o estudo da programação. O bergsonismo se insere no âmbito da filosofia da mente, a qual, por sua vez, está inserida no escopo das ciências cognitivas [Prado, 1999].

1.2.5 Abordagem Bergsonista

O bergsonismo se assemelha, em muitos sentidos, ao método científico. Em outras palavras, sugere-se que ele possa ser considerado uma adaptação do método científico, para ser aplicado aos "fenômenos" da filosofia. Tal qual o método científico, o bergsonismo coloca suas hipóteses com base na intuição, a qual é resultado de um longo período de interação entre o investigador e o objeto.

Naturalmente, a verificação da hipótese deve ser adaptada, uma vez que o objeto de estudo é um "fenômeno" qualitativo. Em outras palavras, enquanto o método científico verifica a hipótese por meio de medições, o bergsonismo recorre mais uma vez à intuição. Nesse contexto, a intuição é usada para supor a duração, isto é, supor uma vivência real, sem deixar que os conceitos que descrevem a hipótese atrapalhem. Seria, mais ou menos, como supor a vivência de um escritor ao ler um poema.

A objetividade, tal qual ocorre com o método científico, se dá de forma intersubjetiva. Ou seja, se inúmeras pessoas reconhecerem, na hipótese, a sua vivência, ela pode ser considerada objetivamente confirmada. Também, tal qual ocorre com o método científico, as hipóteses bergsonistas são passíveis de refinamento, caso seja verificado algum ponto da hipótese que não coincida de forma precisa com a vivência.

Com base nisto, sugere-se a *hipótese de que o bergsonismo possa ser utilizado como abordagem para o estudo da programação de computadores.* E tem o *objetivo de avaliar a possibilidade da aplicação do bergsonismo sobre a programação de computadores.*

Esta pesquisa tem como *escopo a tentativa de aplicar o bergsonismo como abordagem metodológica no estudo da programação*. No entanto, ela não visa apresentar um resultado conclusivo, mas questionar, por meio de uma prova de conceito, se novos estudos neste sentido devem ser realizados ou não. Em outras palavras, este estudo se propõe a descrever o bergsonismo na forma de uma abordagem de investigação e aplicá-lo na produção de uma prova de conceito, capaz de subsidiar uma avaliação preliminar da aplicabilidade do bergsonismo ao estudo da programação.

1.3 Metodologia

Este estudo será desenvolvido tomando o bergsonismo, como descrito no capítulo 3, como abordagem primária de investigação. Isto significa que, neste estudo, o bergsonismo será aplicado recursivamente para avaliar a aplicabilidade do bergsonismo no estudo da programação.

Assim, após ter sido possível desenvolver a intuição referente ao bergsonismo, remover-se-á sistematicamente os símbolos que compõem os textos de Bergson, em busca de sua duração original. Feito isto, se passará a recompor a descrição do bergsonismo mas agora na forma de uma abordagem de investigação. A verificação sobre a adequação da descrição proposta em relação à original se dá no reencontro dos termos e idéias do original na descrição proposta.

Definida a abordagem, ela será utilizada no estudo da programação. A verificação do resultado se dará tanto pelo reencontro ou inferências sobre afirmações em referências bibliográficas, quanto por meio da apresentação de provas de conceito derivadas das vivências do autor.

Naturalmente, a validação, por ser intersubjetiva, cabe ao leitor. No entanto, convém um alerta: tal qual ocorre com o método científico, cuja validação depende da compreensão sobre o seu funcionamento, a validação apropriada de um estudo bergsonista também depende da compreensão sobre o seu funcionamento. Este alerta se deve ao fato do bergsonismo ter sido criticado cada vez que se tentou validar seus resultados, tomando por base um outro paradigma. O fato é que a tentativa de validar os resultados obtidos pelo método científico, tomando como base o bergsonismo, levaria a críticas semelhantes aos resultados obtidos por meio do método científico. Embora não caiba um aprofundamento maior deste assunto nesta seção, ela é melhor discutida no capítulo 5.

1.4 Organização do Texto

O capítulo 2 apresenta um levantamento bibliográfico que revisa o uso do Bergsonismo no estudo da programação. Para isso, discorre sobre as noções de Filosofia da Ciência, Metafísica Computacional e Intuição.

O capítulo 3 apresenta uma revisão conceitual sobre a análise, conforme os princípios do bergsonismo. Para tanto, se discute seu objeto (vivência), elementos principais (extensão, duração e memória). Ele apresenta, então, como tais itens operam durante a uma investigação metafísica e a preocupação de Bergson com a precisão deste método. Com base nisso, sugere-se dois *frameworks*. O primeiro (FIB) volta-se para o estudo da essência do objeto e o segundo (FCB) para a comparação e distinção de objetos metafísicos.

O capítulo 4 aplica o bergsonismo para primeiro separar o misto por meio do FIB e do FCB, e depois o remonta para retomar a experiência de programação. Com base neste entendimento, é possível compreender, com clareza e objetividade, como parece se dar a programação de computadores voltada à construção de programas acadêmicos.

O capítulo 5 apresenta e discute os principais resultados obtidos nos capítulos anteriores. Ele discute a aplicabilidade do bergsonismo no estudo da programação e as principais contribuições desta pesquisa.

Capítulo 2

Revisão Teórica

Existem alguns tipos de conhecimento que são metafísicos por natureza. Por mais que se tente, não é possível estudá-los cientificamente. O bergsonismo é um sistema metafísico, que se propõe a investigar os constructos metafísicos e, sobre eles, construir um tipo de conhecimento semelhante ao da ciência [Bergson, 2006]. Uma interpretação livre disso é que Bergson propôs uma ciência metafísica.

A metafísica difere da ciência, pois enquanto esta considera o ser de forma generalizada e estuda algumas de suas particularidades, a metafísica considera cada ser de forma singular — o estuda holisticamente [Reale, 2002]. Portanto, os objetos metafísicos são aqueles que devem ser estudados de maneira singular e holística. Do contrário, o conhecimento obtido não se mostra útil.

Rittel e Webber propuseram em [Rittel & Webber, 1973] a existência de duas classes de problema: os *tame problems* e os *wicked problems*. De maneira geral, os *wicked problems* não tem uma formulação apropriada e, por isso, não é possível saber quando o problema foi resolvido [Conkling & Christensen, 2009]. Além disso, as soluções encontradas não são certas ou erradas. São mais ou menos adequadas e cada *wicked problem* é novo e singular, além de depender da criatividade individual no vislumbre soluções [Conkling & Christensen, 2009]. Sugere-se, assim, que os *wicked problems* tenham natureza metafísica.

O desenvolvimento de um programa é um *wicked problem* [Vega, 2012-2015]. Não há como saber, *a priori*, se a implementação de uma especificação de requisitos atenderá às necessidades ou metas do *stakeholder* [Marcondes *et al.*, 2009]. Mesmo que se projete uma aplicação com completude interna, não é possível saber se, ao interagir com o ambiente externo, haverão situações imprevistas [Marcondes *et al.*, 2011]. Isso também é evidenciado ao considerar as propriedades essenciais do software, propostas por Brooks em [Brooks, 1995]. A singularidade, por exemplo, é expressa pela propriedade de complexidade; a conformidade consiste na incapacidade de saber se a solução

para um *wicked problem* está certa ou errada; *et cetera*.

Este é um dos possíveis motivos da tentativa de estabelecer uma teoria de programação. Embora seja algo que vem se tentando pelo menos desde a década de 70 [Naur & Randell, 1969], este é um problema ainda em aberto [Johnson *et al.*, 2012]. O fato é que as tentativas nesse sentido tenderam para o âmbito da filosofia. No entanto, como em geral não se adotou nenhum sistema filosófico para dar a sustentação necessária ao estudo, muitas tentativas resultaram em opinião ou metáforas sem utilidade real. Considere, por exemplo disso, algumas tentativas:

> *Fred Brooks says that writing software is like farming, hunting werewolves, or drowning with dinosaurs in a tar pit (1995). David Gries says it's a science (1981). Donald Knuth says it's an art (1998). Watts Humphrey says it's a process (1989). P.J. Plauger and Kent Beck say it's like driving a car (Plauger 1993, Beck 2000). Alistair Cockburn says it's a game (2001). Eric Raymond says it's like a bazaar (2000). Paul Heckel says it's like filming Snow White and the Seven Dwarfs (1994). (...) [McConell says it's like] penmanship or construction.* [McConnell, 2004]

Se diz que tomou uma forma filosófica, pois, segundo Deleuze, a filosofia coloca e pensa conceitos [Deleuze & Guattari, 2010]. E foi essa a abordagem adotada em muitas dessas tentativas. Talvez, o maior problema dessas tentativas tenha sido abordar a questão por uma perspectiva conceitual e recaído no problema da abstração. Grosso modo, a abstração faz com que observe o real em perspectiva e isso leva a um pensamento equivocado, justamente por não se considerar o todo [Teixeira, 2001]. Isso é a base do que Bergson critica sobre a falta de precisão no estudo dos elementos metafísicos [Bergson, 2006].

2.1 Filosofia da Programação

Uma vez que a idéia de filosofia consiste em refletir a respeito de algo por diversas perspectivas, é natural que tais discussões venham permeando todas as áreas de atividade humana. Por isso, pode-se dizer que toda ciência tem uma contraparte filosófica [Turner, 2014]. Na ciência, por exemplo, existe a filosofia da ciência. A filosofia da ciência se ocupa em em refletir sobre o significado dos métodos e resultados obtidos na prática científica. Papineau sugere que a filosofia da ciência deva distinguir claramente a epistemologia e a metafísica da ciência, uma vez que a primeira reflete sobre os métodos e a segunda sobre os resultados e a essência da ciência [Papineau, 1996].

A filosofia da ciência da computação se inspira na filosofia da ciência ao desenvolver suas atividades [Rapaport, 2005]. Ou seja, ela reflete sobre os métodos de computação (epistemologia), sua essência e os resultados (metafísica) [Turner, 2014]. Por exemplo, a epistemologia discute a adequabilidade da análise assintótica, para algoritmos escritos em linguagens de terceira geração. E a metafísica, o significado de um resultado da análise assintótica em linguagens de terceira geração.

Convém ressaltar que o termo "filosofia da computação" é inconsistente com o termo equivalente utilizado no inglês *"philosophy of computer science"* (filosofia da ciência do computador). Esta diferença tem uma origem histórica, onde Dijkstra, em [Dijkstra, 1976], aponta que a programação estava sendo estudada no âmbito da *computer science* (ciência do computador), enquanto, na visão dele, deveria ser inserida em uma área à parte, denominada *computing science* (ciência da computação). Sua idéia era que a ciência do computador deveria se ocupar de pequisas referentes à construção de máquinas concretas (hardware), enquanto a ciência da computação se ocuparia das pesquisas referentes às máquinas abstratas (software) [Dijkstra, 1976].

Embora o termo *computing science* não tenha sido amplamente adotado nos países de língua inglesa, a concepção que o motivava foi. Ou seja, da mesma forma que houve uma variação no uso do termo "ciência", houve também no uso do termo *"computer science"*), que deixou de se referir a computadores e passou a tratar de máquinas abstratas ou virtuais, isto é, de programas. Por isso, o termo *philosophy of computer science* (filosofia da ciência do computador), é traduzido como "filosofia da computação", em português, e trata de questões envolvendo programas. Embora este seja o termo usual, neste estudo, com intenção de reforçar a atenção sobre o objeto de estudos, o termo utilizado será "filosofia da programação", o qual se subdivide em "epistemologia da programação" e "metafísica da programação".

2.2 Intuicionismo

O intuicionismo é uma abordagem reconhecidamente útil no estudo da metafísica [Pombo, 2012]. O intuicionismo toma a intuição como instrumento de investigação. Assim, consegue abordar objetos que não podem ser descritos, apropriadamente, por meio da vivência acumulada e não dos símbolos descritos. A intuição é uma espécie de cognição [Stanovich & West, 2000]. Tal qual a razão, ela está presente em qualquer atividade humana. Isso pode ser percebido ao considerar que mesmo alguns dos maiores expoentes do racionalismo moderno, como Popper [Popper, 2004] e Russell [Russel, 1912], a reconhecem e a apresentam como um conhecimento justificado.

Este conhecimento é o que propicia a interação de uma pessoa com o mundo sem

que ela precise, a todo momento, recorrer à análise. No entanto, do mesmo modo que existe a análise do "senso comum" também existe a intuição do "senso comum". Deste modo, como pode ser dado um tratamento à lógica para que ela se torne rigorosa, é possível dar um tratamento à intuição para que ela se torne precisa. Esta é a motivação do intuicionismo.

Uma vez que a intuição dispensa símbolos, o intuicionismo tem centrando seus esforços em problemas metafísicos.

Isso significa que, uma vez que toda ciência (em sentido *strictu* ou *lato*) tem sua contraparte filosófica [Turner, 2014] e que toda contraparte filosófica de uma ciência pode ser dividida em epistemoslogia e metafísica [Papineau, 1996], logo, o intucionismo pode ser aplicado para tratar as questões metafísicas de qualquer "ciência". Desta forma, o intucionismo, tendo a intuição como método, investiga os problemas metafísicos em diversas áreas de conhecimento. O resultado desta investigação é uma compreensão aprofundada do objeto em questão, a qual pode propiciar uma reforma epistemológica na forma de abordar o objeto em questão.

Embora existam diversas escolas intuicionistas [Pombo, 2012], de forma geral, elas compartilham alguns preceitos. Parece ser consenso de que a intuição seja uma forma de conhecimento precisa, oriunda de inúmeras vivências junto ao objeto em questão. Isso implica que o intuicionismo dá ênfase à essência e dispensa a forma, isto é, que busca a vivência e não a descrição. Além disso, a intuição tem caráter holístico e associativo, se apresentando sempre de maneira expontânea. Em suma, a intuição requer memória e busca a vivência com o real, ao dispensar o uso de símbolos [Bergson, 2006], [Stanovich & West, 2000], [Dreyfus, 1998], [Kant, 1995], [Poincaré, 1905], [Brouwer, 2011], [Hintikka, 2003], *et cetera*.

Não cabe realizar, neste estudo, uma revisão exaustiva do uso do termo intuição ao longo da história, embora convenha caracterizar, mesmo que de maneira superficial, como a filosofia de Bergson difere ou se assemelha à de seus contemporâneos intuicionistas: Husserl e Poincaré. Tal consideração é necessária para justificar a escolha do Bergsonismo em detrimento a estas outras duas escolas.

A filosofia de Bergson coincide em muitos sentidos com a de Husserl [Dupont, 2013], com a exceção de que Bergson propõe uma metafísica [Deleuze, 1999], enquanto Husserl uma fenomenologia [Dupont, 2013]. Isso significa que a filosofia de Husserl se atém à mente, enquanto a filosofia de Bergson se ocupa da realidade. Isso deixa espaço no bergsonismo para a reflexão sobre a natureza das substâncias, o que não ocorre em Husserl. Cabe considerar que, uma vez que o bergsonismo está centrado na idéia de duração, em última instância ele se apresenta como fenomenologia. Embora isso seja verdade, sua intenção metafísica possibilita a reflexão sobre substâncias, desde que

condicionadas à duração de alguém [Bergson, 2006]. Essa possibilidade é indispensável no estudo da programação, onde ela só faz sentido quando realizada na forma de um programa, o qual interage com a realidade. Neste sentido, a fenomenologia foge deste escopo.

A semelhança entre as idéias de Bergson e Poincaré pode ser vista em [Flammarion, 1913]. Um registro de que ambos participaram de, pelo menos, uma conferência juntos refletindo sobre o materialismo e o papel que este estava desempenhando da sociedade. Bergson, em [Bergson, 1910], trabalha dois tipos de intuição: a de extensão e a da duração e a intuição da extensão é essencialmente a intuição de Poincaré [Poincaré, 1907]. O fato é que, embora reconhecesse a extensão como algo importante, ele não se interessou em desenvolvê-la. Poincaré, por outro lado, centrou seus esforços em desenvolver justamente este tipo de intuição. Em programação, a extensão é também algo essencial e não pode ser dispensada, por isso, neste estudo, as idéias de Bergson serão complementadas com as de Poincaré, sempre que for necessário.

Por exemplo, a investigação metafísica de Poincaré [Poincaré, 1905] e [Poincaré, 1907], sobre o raciocínio matemático, contribuiu para que Brouwer desenvolvesse um paradigma epistemológico alternativo para a matemática, chamado "intuicionismo". O intucionismo matemático (ou brouwerismo) tem como base dois decretos. O primeiro, determina que a matemática é um fenômeno cognitivo que prescinde de linguagem e tem origem na percepção temporal, continuamente retida pela memória que se tem ao dar espaço ao próximo fenômeno cognitivo [da Costa, 1992]. O segundo, determina que um novo objeto matemático pode ser ou uma entidade definida por meio de suas propriedades, ou uma sequência de escolhas sobre entidades. Portanto, cada criação na matemática brouwerista, seja entidade ou sequência, deve ser passível de construção e ser oriunda de uma intuição numérica, *a priori* [da Costa, 1992]. A partir desses decretos, Brouwer desenvolve toda uma concepção matemática alternativa [Heyting, 1971].

Para Bergson, a razão e a intuição são complementares [Bergson, 2006]. A diferença é que a razão opera sobre a extensão e a intuição sobre a duração. Em programação, a intuição, por apreender o real, valida se as construções da razão fazem sentido no mundo fenomênico, enquanto a razão, por ser abstrata (ou sígnica), verifica logicamente as construções da intuição *cf.* [Jackson, 2013].

2.2.1 Intuicionismo na Programação

De forma semelhante, se acredita que o tratamento intuicionista dos problemas apresentados pela metafísica da programação, possa trazer alguns esclarecimentos que propiciem o refinamento epistemológico da programação e, deste modo, tenham algum

chance de sucesso em atacar dificuldades, como as apresentadas pelo SEMAT.

A intuição também é amplamente reconhecida na programação, embora seja tratada de maneira informal. Naur apontou, em 1984, a necessidade do tratamento da intuição no desenvolvimento de software [Naur, 1985]. Michael Jackson também apontou neste sentido, em 2013 [Jackson, 2013]. Nesse ínterim, diversos outros pesquisadores também fizeram menção à importância do uso da intuição na programação, embora a idéia não tenha sido melhor desenvolvida [Beynon *et al.*, 2008].

Outrossim, o uso da intuição nesses artigos era a intuição voltada ao desenvolvimento de aplicações e não ao estudo de questões metafísicas, como proposto neste estudo. Naturalmente, o uso apropriado da intuição, durante a programação, pode trazer um ganho de qualidade ao programa. No entanto, o fato de isto nunca ter sido aprofundado [Beynon *et al.*, 2008] sugere que ainda não foi possível entender como fazer isso. Essa questão, de qual o papel da intuição durante a programação, é uma questão metafísica. Portanto, antes de elaborar como usar a intuição na produção de software (epistemologia), é preciso entender como a intuição auxilia na criação de linhas de código de um programa (metafísica). Neste sentido, convém usar o intuicionismo para estudar as questões metafísicas da programação.

Dentre as abordagens intuicionistas consideradas, foi o bergsonismo que se mostrou mais adequado aos objetivos deste estudo. O bergsonismo é uma escola intuicionista, que se propõe a desenvolver um método de investigação metafísica que propicie resultados tão precisos quanto os da ciência. Tal propriedade é de interesse uma vez que, por um lado, se deseja desenvolver uma ciência da programação e, por outro, a programação é um elemento metafísico.

2.3 Bergsonismo na Programação

Existem alguns poucos artigos na computação que fazem citação direta a Bergson, ou o usam como referência. São 31 na base de artigos da ACM (*Association for Computing Machinery*) com a chave "Henri Bergson"). Entre eles, alguns tratam de arte, como [Voegelin, 2006] e [Guy & Champagnat, 2012], e outros de métodos de computação alternativos, como [Resconi & Nikravesh, 2008] e [Øhrstrøm, 2010]. Possivelmente, a maioria dos textos tratam de questões metafísicas, mas no sentido filosófico como [Cohen, 2000], [Haynes, 2001], [Robbins, 2002] e [Cerqui, 2002].

Um estudo que cabe ressaltar é o de Hansen em [Hansen, 2004]. Ele usa o par Bergson e Deleuze para estudar as novas mídias. Pode-se dizer que seja um prolongamento do esforço de Deleuze de pensar o cinema sob a ótica bergsonista *cf.* [Deleuze, 1986]

aplicado à temas como realidade virtual e inteligência artificial. No entanto seu foco é a reflexão sobre a interação entre pessoas e tecnologias na formação dos chamados "pós-humanos"[Hansen, 2004]. Pós-humano é uma concepção filosófica de que a tecnologia computacional possa ser usada para ampliar as capacidades humanas. Grosso modo, o estudo pós-humano reflete sobre as possibilidades e consequências da ampliação e mediação artificial e mecânica da capacidade humana. Assim, Hansen trás o bergsonismo para estudar o "uso" da tecnologia computacional enquanto este estudo trás o bergsonismo como método para estudar a criação da tecnologia computacional.

O único trabalho encontrado que faz referência a Bergson, com a preocupação de desenvolvimento de software, foi [Marcondes *et al.*, 2009]. No entanto, esse estudo, um *position paper*, se limitou a usar o Bergson para apresentar um problema da programação, mas sem oferecer nenhuma solução para o problema apresentado.

2.3.1 *Bergson-Hume forms*

O único trabalho encontrado, com tema correlato a este, foi [Miyoshi, 2004]. Esse trabalho sugere que a computação possa ser estudada com base em um *framework* denominado *Bergson-Hume forms* (BHF), apresentado na figura 2.1. Miyoshi entende que duração e extensão sejam complementares, por isso escolhe Hume para tratar o lado da extensão e, com isso, complementar o lado duração tratado por Bergson.

Tal escolha se deve, possivelmente, ao fato de Miyoshi considerar as idéias de Hume compatíveis com as idéias de Bergson. Em especial, na perspectiva da filosofia da mente. Assim, ele usa a causalidade de Hume para compor o modelo [Miyoshi, 2004]. A escolha por Hume, implica na intenção de investigar a computação pela forma com a qual ela é percebida [Morris & Brown, 2014]. Isso torna o BHF adequado para o estudo metafísico da computação [Miyoshi, 2004].

A principal crítica, a ser feita sobre este estudo, é que ele aplica o BHF sobre entidades ou fenômenos [Miyoshi, 2004]. Pode-se definir duração como uma preocupação no tempo, mas quem atribui essas qualidades é o ser humano [Bergson, 2006]. Isso significa que uma entidade, enquanto ser sem consciência, não pode ter duração. No entanto, o ser humano, ao interagir com o objeto, pode lhe atribuir uma qualidade conforme a duração do humano.

Considere, por exemplo, um bloco de madeira. Um bloco de madeira não tem uma vida interior que o permita ter duração. No entanto, o ser humano, ao observar o bloco de madeira, pode lhe atribuir a qualidade de escada (caso sua preocupação seja pegar algo em um lugar algo), cadeira (caso sua preocupação seja descansar), adorno (caso

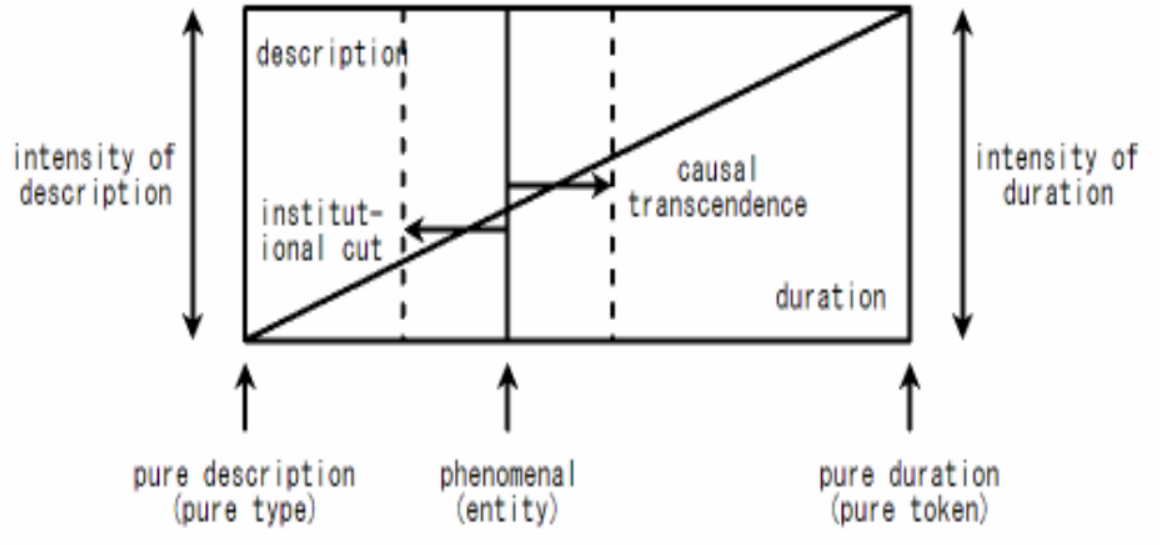

Figura 2.1: *Bergson-Hume forms cf.* [Miyoshi, 2004].

sua preocupação seja artística), lixo (caso sua preocupação seja organizar), *et cetera*. Sempre conforme a sua duração.

De forma geral, as divergências entre o BHF e o *framework*, proposto neste estudo, tem origem nesta divergência de interpretação do objeto de estudos do bergsonismo.

2.4 Descrição do Misto da Programação

Para Bergson, a realidade se apresenta como um misto, ou seja, o misto é própria realidade [Deleuze, 1999]. Em outras palavras a realidade consiste na mistura de diferentes coisas em diversos graus e várias qualidades. Tomemos por exemplo um ambiente qualquer; ao observá-lo, se percebe uma mistura das mais variadas coisas que podem ser percebidas em diferentes níveis de abstração e interesse, conforme o "estado de espírito" da pessoa no momento. Portanto, é natural que a programação esteja inserida no misto do real. A figura 2.2 apresenta um diagrama de um quadro conceitual, que pode ser usado para delinear o misto no qual a programação está inserida. Naturalmente, a figura 2.2 não é a realidade, mas uma representação ou inspiração de um universo de discurso.

Problem-Frame Extendido

O quadro conceitual apresentado tem, como base, o *problem-frame* proposto por Jackson em [Jackson, 2005]. O *problem-frame* consiste em uma especificação de requisitos, que expressa uma necessidade de impelir o mundo do problema (contexto no qual o

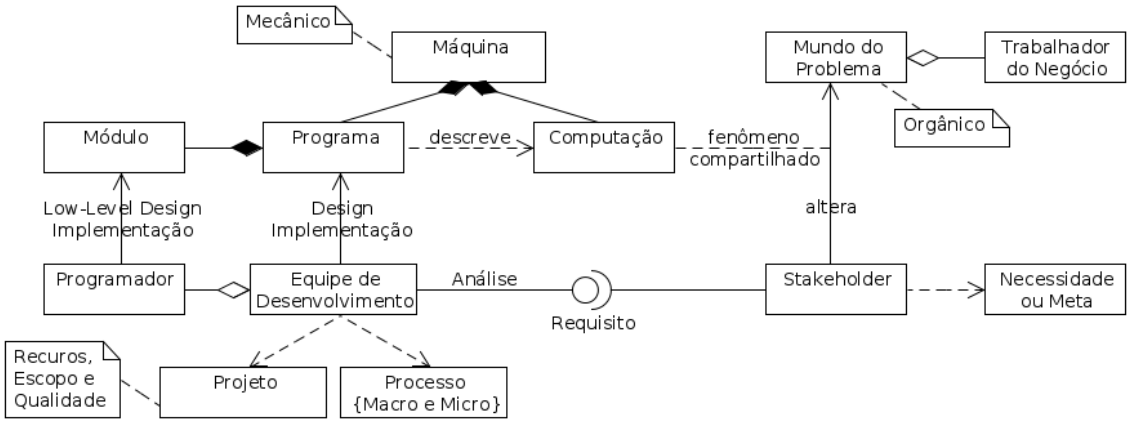

Figura 2.2: Diagrama Conceitual em UML do *problem-frame* de Jackson-Vega.

sistema será inserido) à uma condição de interesse. Para isso, é concebida uma máquina [virtual], que, por meio de sua interação com o mundo do problema, deve realizar os requisitos, na esperança de que a necessidade de estimular uma nova condição ao mundo do problema seja atendida.

O *problem-frame* foi, de certa forma, extendido por Vega, de forma que este passou a incluir também a preocupação com o projeto do *problem-frame* [Vega, 2012-2015]. Para Vega, a necessidade de impelir o mundo do problema a uma condição de interesse é reflexo de uma necessidade ou meta de negócio, que o *stakeholder* deseja atender ou atingir. Ao ser contratada, a equipe de desenvolvimento, aplicando alguma variação sobre o método de *problem-solving* de Polya, busca produzir a descrição de um cálculo que, ao ser realizada por uma máquina, apresente o funcionamento determinado pelos requisitos [Polya, 1957].

Com relação à forma de interpretar os requisitos, Jackson os interpreta como a descrição de uma condição a ser atingida pelo mundo do problema, ao interagir com a máquina [Jackson, 2005]. Vega, por outro lado, os interpreta como funções-λ, as quais descrevem o funcionamento a ser apresentado pela máquina [Vega, 2012-2015]. Nesta perspectiva, é o quadro de Jackson que possibilita maior abrangência, pois entende que a computação não pode ser nominalista e que o funcionamento da máquina deve ter efeitos concretos no mundo do problema, para considerar que a computação foi bem sucedida.

Problem-Solving

Polya entende como *problem-solving* o uso consciente de um conjunto de técnicas ou heurísticas, visando a solução de um problema [Vega, 2012-2015]. Problema pode ser

entendido como um desajuste entre a condição real e uma condição de interesse do mundo do problema [Johns & Saks, 2004]. A estratégia de *problem-solving*, proposta por Polya, consiste em um conjunto de fases (ou macro-processo *cf.* [Booch *et al.*, 2007]) e heurísticas (ou micro-processo *cf.* [Booch *et al.*, 2007]), das quais se espera que, quando devidamente aplicadas, propicie que o mundo do problema assuma uma condição de interesse [Polya, 1957].

Ao considerar que a necessidade de impelir uma mudança no mundo do problema tem origem em uma necessidade ou meta de negócio, naturalmente outras preocupações se tornam parte do problema.

Entre elas, está a preocupação com os aspectos de projeto (escopo, qualidade e recursos) em relação à oportunidade de negócio e aos riscos de implantação [Boehm & Sullivan, 2000]. Em outros termos, com este refinamento de Vega, é possível considerar todo o planejamento estratégico de TI e associá-lo a cada computação descrita ou realizada [Vega, 2012-2015]. A extensão de Vega propicia também que os aspectos relativos ao *problem-solving* possam ser considerados. Esta preocupação propicia considerar, no estudo da computação, os aspectos relativos tanto aos macro e micro-processo de [Booch *et al.*, 2007], como aos sócio-cognitivos de [Weinberg, 1998].

Computação Interativa

Entre a máquina (ou computação) e o mundo do problema ocorre o que Jackson chamou de fenômeno compartilhado. O fenômeno compartilhado representa a interrelação entre o dispositivo computacional e o operador [Jackson, 2005]. Por um lado, *stakeholder* espera fazer com que o mundo do problema atinja uma condição de interesse por meio dele [Jackson, 2005]. Por outro lado, dado que o mundo do problema é um sistema orgânico e a máquina um sistema mecânico, é natural que o primeiro exija do segundo regulagens constantes [Bertalanffy, 2008]. Além disso, o fenômeno compartilhado implica que a máquina seja uma Máquina de Turing do escolha (*choice-machine cf.* [Turing, 1937]) [Vega, 2012-2015].

Embora Turing nunca tenha detalhado o funcionamento desta máquina, ele a apresenta como uma máquina automática não determinista, onde a indeterminação é resolvida pelo operador [Turing, 1937]. Como exemplo, considere a função de transição parcial não-determinista $(q_0, a) \rightarrow \{(q_1, b, D)|(q_2, c, E)\}$. A máquina de escolha para e aguarda uma decisão do operador. Ao elaborar esta propriedade, torna-se possível ao operador escrever na fita enquanto a máquina aguarda o evento. Pode-se dizer que, enquanto a máquina automática formaliza a noção de algoritmo, a máquina de escolha formaliza a noção de dedução.

A dedução ocorre pela aplicação sucessiva de regras de inferência sobre premissas ou axiomas. Como não existe uma norma para a aplicação das regras de inferência, elas são escolhidas a cada momento, de forma arbitrária, caracterizando um procedimento reativo (responde a eventos e à passagem de tempo) [Harel, 1987]. A participação de um operador durante computação torna a máquina de escolha mais poderosa, computacionalmente, do que a máquina automática. Ou seja, a torna um constructo matemático mais geral do que a máquina automática [Wegner, 1997], ao propiciar a computação de sistema axiomáticos [Turing, 1937].

Capítulo 3

Fundamentos da Investigação Bergsonista

A obra de Bergson buscou explorar os diferentes aspectos do que ocorre na vida interior das pessoas. Entre outros assuntos ele trabalhou a questão do riso em [Bergson, 1980], da criatividade [Bergson, 1922], da consciência [Bergson, 1920], *et cetera*. Tais estudos foram efetuados tomando como base o *método da intuição* [Bergson, 2006].

A abordagem analítica "tradicional" seria tomar como base o objeto metafísico e tentar conceitualmente separá-lo em partes. O problema disso, é que a demarcação, no fim, será arbitrária e cada pessoa que tentar fazer a mesma coisa, mesmo que parta deste mesmo objeto, possivelmente chegará em resultados diferentes. Como apresentado no capítulo 2 esta foi a abordagem utilizada nas diversas tentativas realizadas no sentido de criar uma teoria de programação.

A análise bergsonista rejeita, inicialmente, todo conhecimento que se tem sobre o objeto deixando para reencontra-los *a posteriori*. Desta forma, a caracterização parte da vivência em si. No bergsonismo, ao se colocar em uma vivência por meio de um esforço de intuição, é possível tentar identificar a extensão e a duração (divergência de tendências) bem como a memória (convergência de tendências). Apenas depois disso ter sido alcançado, é que se pode reinserir o objeto no misto a que ele pertence para entender sua interação com ele. A validação do resultado se dá objetivamente pelo reconhecimento de uma descrição na própria experiência do avaliador.

Para que esta análise seja realizada de maneira estruturada, este estudo propõe a estruturação dos aspectos de investigação do bergsonismo na forma dos *frameworks* apresentados durante este capítulo.

3.1 Elementos do Bergsonismo

O objeto de estudo do bergsonismo é a *vivência*, isto é, aquilo que se passa no interior ou no "espírito" de uma pessoa [Bergson, 2006]. Bergson acredita que a vivência é um misto de tendências que ora diverge e ora converge [Deleuze, 1999]. Quando as elas divergem, a vivência se apresenta como *duração* e extensão [Bergson, 1910], quando convergem ela se consolida na forma de *memória* [Bergson, 1911] e quando evolui suscita *impulso vital* [Bergson, 1922]. É o entendimento desta dinâmica que propicia o método de investigação metafísica e a precisão almejada por Bergson [Bergson, 2006].

> Duração (*Durée*), Memória (*Mémoire*) e Impulso vital (*Élan vital*) marcam as grandes etapas da filosofia bergsoniana. (...) Parece-nos que a Duração define essencialmente uma multiplicidade virtual (*o que difere por natureza*). A Memória aparece, então, como a coexistência de todos os *graus de diferença* nessa multiplicidade, nessa virtualidade. Finalmente, o Impulso vital designa a atualização desse virtual segundo *linhas de diferenciação* que se correspondem com os graus — até essa linha precisa do homem, na qual o Impulso vital toma consciência de sí. [Deleuze, 1999]

3.1.1 Vivência

A vivência é o objeto do bergsonismo pois, na visão de Bergson, não há nada que se possa conhecer mais do que a si mesmo [Bergson, 1911] e [Bergson, 1922]. Este profundo conhecimento sobre si, isto é, este vasto acúmulo de memória sobre si é o que possibilita o uso da intuição sobre si. A suposição de uma vivência de outra pessoa consiste em um processo de empatia o qual permite a pessoa se colocar no lugar da outra e na medida do possível, compreender sua vivência [Deleuze, 1999].

A suposição da vivência de um objeto é impossível ,pois a pessoa precisaria ter vivenciado ser aquele algo antes que pudesse supor sua duração. Sem esta vivência, resta fazer uso da imaginação e o resultado disso foge da proposta do bergsonismo. Embora não seja possível supor a vivência de um objeto, a pessoa, dentro das necessidades da sua vivência pode atribuir uma duração ao objeto. No entanto, a duração do objeto está condicionada à do ser humano, de forma que não se trata da duração do objeto propriamente dita, mas da vivência de uma pessoa.

Tanto melhor será a suposição da vivência de outrem quanto maior for o acúmulo de memória na vivência de algo semelhante. Considere, por exemplo, um programador que busca compreender o motivo de outro programador ter escrito um programa de determinada maneira. Certamente, sua suposição será mais próxima do real do que

a de uma pessoa que nunca programou tentando fazer o mesmo. No entanto, ainda existem nuanças, por exemplo, um programador acostumado a desenvolver programas pequenos (10kloc) pode não compreender as escolhas de um programador de sistemas colossais [Weinberg, 1998].

Para Bergson, a vivência é o real [Bergson, 1911] e o real se apresenta como um misto [Bergson, 1910]. Em outras palavras a realidade consiste na mistura de diferentes coisas em diversos graus e várias qualidades [Deleuze, 1999]. Considere por exemplo um ambiente qualquer, ao observá-lo percebe-se uma mistura das mais variadas coisas que podem ser percebidas em diferentes níveis de abstração e interesse conforme o "estado de espírito" da pessoa no momento. No caso de um programa, por exemplo, conforme as decisões vão sendo tomadas e passam a compor o código, dificilmente se consegue dissociá-las *à postéirori* [Jacobson, 1995].

Assim, o misto, isto é, a vivência é algo homogêneo e por isso não pode ser descrito em sua totalidade e por isso requer perspectivas [Bergson, 2006]. Cada perspectiva apresenta o misto por um ângulo, com intenção e nível de percepção diferentes. Embora cada uma dessas perspectivas tenha origem em uma mesma experiência, quando reunidas elas não formam um todo coeso que representa a experiência original. No entanto, quanto mais descrições houverem maior será o subsídio para que se suponha a vivência que os originou. Em programação esta situação pode ser apresentada como a diferença entre o modelo (mental) e os diagramas (UML) que o representam [Peirce, 2012].

3.1.2 Duração

A idéia de duração, como o próprio nome diz, é uma referência ao tempo. No entanto Bergson critica o hábito de se contar o tempo em termos quantitativos, enquanto este se mostra uma vivência qualitativa [Bergson, 1910]. Essa tese de Bergson pode ser compreendida ao considerar que existem situações que uma mesma quantidade de tempo passa mais rápido ou mais devagar conforme aquilo que se está vivenciando.

Bergson utiliza esta idéia para sugerir que no tempo existam duas tendências, uma é a do tempo quantitativo e a outra do tempo qualitativo [Bergson, 1910]. O tempo quantitativo recebe o nome de extensão e o qualitativo de duração [Deleuze, 1999]. Esta separação em extensão e duração, ou em outros termos, em espaço e tempo sugere que Bergson aceita o tempo como algo qualitativo e a extensão como algo quantitativo. De maneira mais rigorosa, Bergson entende que tanto a extensão como a duração sejam multiplicidades [Bergson, 1910]. A multiplicidade na extensão forma uma sucessão de pontos como ocorre na física. A multiplicidade da duração é composta por estados

mentais que duram durante a vivência.

Daí que para Bergson, a duração é a própria vivência enquanto a extensão e a matéria com que se interage durante a vivência [Bergson, 1910]. Em termos aristotélicos, no bergsonismo, a essência está na duração e o acidente na extensão [Deleuze, 1999]. Sendo assim, a duração é uma continuidade, um fluxo, que se estende por toda a vida de uma pessoa enquanto a extensão se dá na forma discreta de momentos entremeados com eventos. Embora a duração seja resultado de uma continuidade, em cada duração a mente fica absorta na vivência [Bergson, 2006] por todo o tempo que esta durar e a continuidade se dá em termos da memória [Bergson, 1911].

É possível afirmar que enquanto durar uma vivência, a duração se apresenta como uma multiplicidade de nuanças [Deleuze, 1999]. Essas nuanças são fruto de estímulos tanto do ambiente quanto da cognição que direcionam a mente em um determinado sentido. Suponha que alguém esteja dirigindo um carro, embora a vivência de dirigir o carro se mantenha por todo trajeto, diferentes durações ou nuanças de duração podem se interpor. Se diz que houve mudança na duração quando existe uma diferença de natureza entre uma duração e outra; se diz que houve mudança na nuança da duração quando não existe mudança de natureza entre uma nuança e outra [Bergson, 2006].

Na vivência de programação, por exemplo, existe uma diferença de duração (natureza) entre as fases de análise e implementação mas uma diferenças de nuança entre a modelagem e a codificação. Grosso modo, durante a análise, o programador tem a mente voltada para o entendimento de uma realidade enquanto na implementação, ela se volta para a concepção de um comportamento, daí a mudança na duração. Por outro lado, tanto a modelagem como a codificação permanecem com a mesma preocupação de um conceber um comportamento seja com um nível de detalhe e rigor maior (codificação) ou menor (modelagem). Este assunto será detalhado nos capítulos subsequentes.

3.1.3 Memória

Embora a duração seja o resultado de uma continuidade [Bergson, 1910], quem lhe confere esta propriedade é a memória por prolongar os eventos passado no presente [Bergson, 1911] como coexitência virtual [Deleuze, 1999]. Em outros termos é possível afirmar que extensão e duração convergem na formação de memória, ou ainda, que a memória é o resultado da vivência.

Para Bergson existem duas formas de memória, a memória mecânica e a lembrança [Bergson, 1911]. A memória mecânica é a memória do hábito que se alcança por meio do treino e uma vez obtida ela se torna automática, é a memória utilizada para dirigir

um carro. A lembrança é a memória referente à duração de vivências e ela é acumulada como o misto da realidade. A memória mecânica é uma memória utilitária e não lembra de eventos específicos apenas do movimento que deve ser realizado; a lembrança é uma memória reflexiva que permite retomar toda uma vivência [Bergson, 1911].

Em programação, é a memória mecânica que traz produtividade por permitir a redação "automática" de um conjunto grande de instruções já conhecidas. Em geral tais instruções são referentes à interfaces com tecnologias conhecidas ou resultantes de um processo de inferência simples como a da descrição de regras de negócio. A lembrança, por outro lado, é a memória que ao prolongar-se propicia que se reconheça vivências passadas [Deleuze, 1999], possibilitando assim uma reflexão relativa às decisões de projeto.

Considere como exemplo a preparação de um prato por um *chef de cuisine*. Existe toda uma vivência envolvida na preparação do prato onde se interpõe diferentes preocupações. O resultado da forma com que o *chef* tratou essas preocupações, incidentes que ocorreram e outras intercorrências ficam consolidadas no prato. Então, por um lado, não se pode dizer que o prato é a vivência pois a vivência foi prepará-lo, por outro lado, não se consegue dissociá-lo da vivência pois ele é o substrato sobre o qual se deu a vivência. Por consolidar a vivência é que o prato pode ser considerado memória e por ser uma forma é que se pode dizer que ele representa o lado da extensão.

3.1.4 Impulso Vital

No bergsonismo o impulso vital é o mecanismo que pelo qual a vivência se desdobra e se dissocia [Bergson, 1922]. Durante a vivência se manifestam uma multiplicidade de virtuais, dentre eles somente alguns se atualizam. Um virtual é uma vivência em potencial, a qual, precisa atualizar-se para compor a vivência atual [Deleuze, 1999]. A atualização de um virtual requer um esforço, isto é, um ímpeto que o faz a duração "mudar sua direção" tornando-se outra. Este ímpeto na realização do esforço de atualização é o que Bergson chama de impulso vital [Bergson, 1922].

Para Bergson, atualização equivale a criação de diferenças e isso se deve à memória [Deleuze, 1999]. A memória é o elemento que se prolonga entre as durações, isso faz com que cada duração seja nova em função da memória da duração passada [Deleuze, 1999]. Em outros termos, a evolução criadora de Bergson se dá por tentativa acumulada em face de problemas específicos e dos meios que se tem para solucioná-los [Bergson, 1922]. Assim, embora existam diferenças entre as tentativas elas serão sempre as melhores possíveis dentro daquilo que é possível [Deleuze, 1999].

Considere como exemplo o caso de dois programadores que receberam uma mesma

especificação técnica. Em face deste problema, cada um deles fará uso de suas vivências de programação anteriores (memória) para apresentar a melhor solução possível. Caso ambos programadores recebam a mesma especificação técnica uma segunda vez, a memória já contará com a vivência de ter feito este programa uma primeira vez então o programa resultante não será igual ao primeiro. Com a repetição deste processo é possível conseguir versões cada vez mais refinadas do programa em questão. No entanto, com o tempo esta evolução tende a estabilizar e assim os programas começam a ser produzidos de maneira cada vez mais semelhante (embora nunca idêntica).

A evolução criadora estabiliza pelo esgotamento dos meios que se tem para resolver um problema. Assim, caso o programador tenha contato com ferramentas, técnicas ou princípios novos, a evolução criadora é retomada até que novamente se estabilize. Isso implica que cada refinamento requer um impulso vital que suscita um esforço inteiramente novo na solução de um problema já solucionado. O mesmo processo ocorre na criação de qualquer coisa onde a idéia é um virtual que, por impulso vital, se atualiza na forma de extensão [Bergson, 1922].

Em suma, um virtual é uma construção mental em potencial, que é continuamente apresentada à mente conforme seu direcionamento [Deleuze, 1996]. Dado um virtual, um primeiro impulso vital é atualizá-lo na mente, formando o que [Brouwer, 2011] chamou de construção mental. Esta atualização consiste em conceituar um virtual inicialmente destituído de símbolos, para que ele se torne inteligível. Um segundo impulso vital consiste em buscar formas para atualizar, na matéria, o virtual manifesto como construção mental. Neste sentido, a construção mental ou a construção material, consolida a atualização de um misto com virtuais de diferentes naturezas, formando o que [Bergson, 2006] intitula memória. [Bergson, 2006] chama de extensão os conceitos ou a matéria pela qual se atualiza um virtual. Cabe frisar que a atualização de um virtual é opcional. Isso significa que um virtual pode nunca se tornar uma construção mental ou que uma construção mental pode nunca se materializar no real.

3.1.5 Precisão

O bergsonismo foi proposto com a intenção de fazer da filosofia uma área de conhecimento tão precisa em seu escopo, quanto a ciência é no dela. Bergson entende que a precisão de uma teoria depende do quanto ela se refere a apenas a um único objeto e faz sentido apenas em seu contexto [Bergson, 2006]. Isso é similar à proposta de Popper em [Popper, 2004], de que a ciência deve ser dedutiva e evitar generalizações, para ser precisa.

Um método que se supõe preciso, deve produzir teorias passíveis de validação. Para

que a validação seja possível, a teoria deve objetiva (ou intersubjetiva). Para Bergson a evidência filosófica é diferente da científica por esta, muitas vezes, não ser observável e nem mensurável, e sim qualitativa ou cognitiva [Bergson, 2006]. Desta forma, um especialista ao ler a teoria deve imediatamente reconhecê-la como verdadeira, isso é o que caracteriza a objetividade filosófica do bergsonismo [Deleuze, 1999].

A objetividade filosófica do bergsonismo propicia, assim como ocorre na ciência, o desenvolvimento incremental onde as teorias se tornam passíveis de constante refinamento, evolução e refutação na busca pelo absoluto. Absoluto para Bergson não tem conotação de imutável mas da expressão pura e precisa da experiência com o real [Bergson, 2006]. Assim, tal qual ocorre com a observação científica, não se deve esperar nada de radicalmente novo em uma teoria proposta pelo bergsonismo mas apenas a constatação do óbvio em termos da consciência. Pode-se esperar um conhecimento cada vez mais preciso e sistemático de uma experiência com o real.

3.2 Método de Investigação

A base da investigação bergsonista é a intuição [Deleuze, 1999]. No entanto Bergson foge de tentar formular uma definição simples de intuição [Bergson, 2006]. Mesmo Deleuze se abstém deste intento [Deleuze, 1999]. Talvez isso se deva ao fato de intuição ter natureza metafísica e portanto uma formulação simples não pode ser dada mas apenas tangenciada. Intuição, tal qual a razão é uma forma de conhecimento [Stanovich & West, 2000] e ao buscar uma formulação para razão se encontra esta mesma dificuldade. Portanto, da mesma forma que o entendimento intuitivo sobre a razão é suficiente para falar sobre ela, neste estudo, considerar-se-a também suficiente o conhecimento intuitivo que se tem de intuição.

> A *intuição* é o método do bergsonismo. (...) Trata-se de um método essencialmente *problematizante* (crítica de falsos problemas e invenção de verdadeiros), *diferenciante* (cortes e intersecções), e *temporalizante* (pensar em termos de duração). [Deleuze, 1999]

A investigação bergsonista se dá por meio da tomada de contato direto com o objeto sem deixar que nada se interponha [Bergson, 2006]. Isso significa que é necessário rejeitar tudo aquilo que é abstrato (símbolos, conceitos, métricas, *et cetera*) para, por meio de um esforço de intuição, retomar uma vivência em particular. Esta vivência não se trata de uma vivência em geral mas de uma vivência específica a qual se apresenta à consciência de forma homogênea e única. Em outros termos, o bergsonismo subverte o hábito do pensamento conceitual e por isso pode parecer exótico a primeira vista.

Uma vez acessada a vivência é possível "observar" como ela diverge em termos de extensão e duração e como ela converge em termos de memória. Na divergência é possível identificar os elementos que compõe o objeto bem como diferenciar durações com natureza diferente das durações com mesma natureza (nuanças). Na convergência é possível identificar como tais elementos se relacionam na construção da memória e como as multiplicidades se consolidam formando a extensão e a duração. Para isso, o bergsonismo empreende um esforço inteiramente novo em cada investigação

A divergência bergsonista se dá em dois níveis [Deleuze, 1999]. O primeiro nível divide a vivência em termos de extensão (1) e duração (2). O segundo nível divide a duração também em termos de extensão (2) e duração (3). Neste caso, a extensão representa a variação simbólica ao longo do tempo; a duração, sendo indivisível, se mantém igual. Para Bergson, apenas o (3) é absoluto pois ele sempre é reencontrado mesmo em meio à multiplicidade de expressões que a representam e é a partir dele que se reencontram os conceitos que irão expressar o (2) [Deleuze, 1999]. Tais direcionamentos de análise formam uma espécie de *framework* como apresentado na figura 3.1.

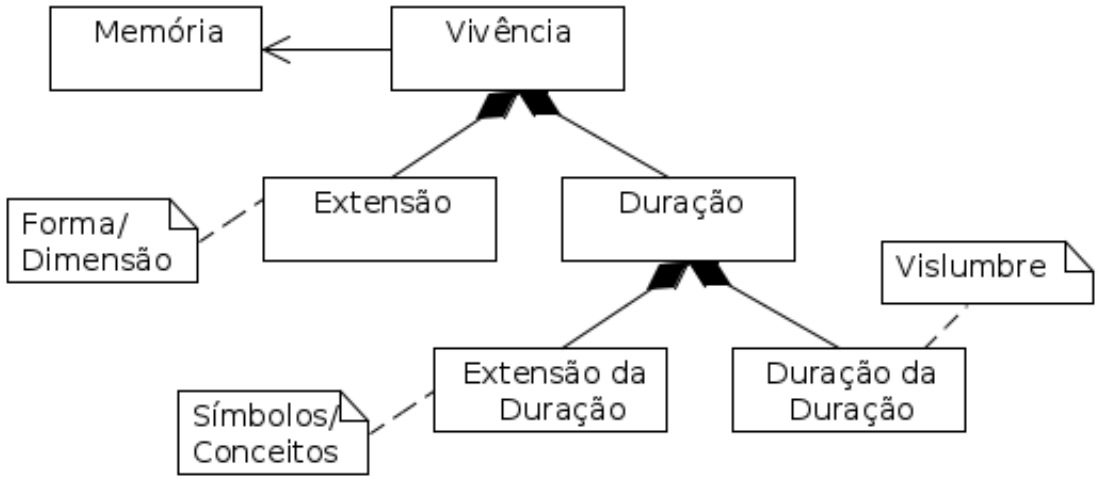

Figura 3.1: Diagrama UML do *Framework* de Investigação Bergsonista (FIB).

A duração da duração é algo que suscita vislumbres os quais, por serem virtuais, se atualizam por meio dos símbolos formando um constructo mental, estes por sua vez, se concretizam por meio de uma forma na matéria [Bergson, 2006]. Por exemplo, a duração da duração pode suscitar um vislumbre de um número o qual se atualiza por meio de um símbolo que o representa, no entanto, este só se concretiza quando a forma que o representa é escrita no papel. Essas passagens de atualização e concretização é o que Bergson chamou como impulso vital [Deleuze, 1999].

Uma vez que se efetue este mesmo procedimento para diversas vivências é possível encontrar padrões de consciência onde esta se "molda" de forma apropriada para cada tipo de situação. Com isso é possível identificar tipos de "montagens" cognitivas que variam conforme o sistema projetivo. Isso, por sua vez, propicia uma classificação qualitativa dos sistemas projetivos propiciando entender, por exemplo, se programação é ou não arte ou é ou não engenharia. Pode-se sugerir *a priori* que assim como arte é arte e engenharia é engenharia, programação não é nem arte e nem engenharia mas que programação seja programação.

Durante todo este processo se começa a reencontrar ou criar abstrações que representam de forma apropriada os elementos identificados na duração. Com base neste reencontro é que se torna possível descrever a experiência. Porém tal descrição seja talvez a maior dificuldade a ser superada no bergsonismo. A investigação bergsonista resulta em uma compreensão homogênea e holística de uma realidade, por outro lado, a descrição da vivência, devido à limitações da linguagem, precisa ser discreta e apresentada em perspectiva. Equacionar essas diferenças visando apresentar uma descrição de forma organizada não é algo trivial e possivelmente foi isso que suscitou muitas das diversas críticas ao bergsonismo, como por exemplo a de Russell em [Russel, 1912].

O conhecimento obtido por meio do bergsonismo, assim como o do método científico, é incremental e passível constante refinamento, evolução e refutação. Outrossim, se espera que um método com esta proposta propicie a duas pessoas acessarem uma mesma duração, levando-as a conclusões bastante próximas, tal qual ocorre na observação científica. É neste sentido que [Bergson, 2006] fala em precisão filosófica, e apresenta a intuição como o método da investigação filosófica.

A validação no bergsonismo se dá por meio da reconstrução do real, *i.e.*, o resultado da intuição deve coincidir com a experiência, permitir experimentos intelectuais ou mesmo propiciar a busca de evidências no real. A evidência filosófica é diferente da científica por esta, muitas vezes, não ser observável e nem mensurável, mas qualitativa ou cognitiva. A verificação no bergsonismo se dá pela possibilidade de diferentes pessoas, acessando a duração do autor, reconstruir sua duração e ver, em sua descrição, a única, ou ao menos, a melhor possibilidade com relação à experiência real. Caso se verifique não ser este o caso, o estudo pode então ser refutado e uma nova teoria ser colocada em seu lugar.

O rigor do bergsonismo fica evidente quando não se consegue recompor de maneira racional os elementos de uma vivência. O resultado disso é semelhante à forma que se manifesta o rigor de uma prova matemática, onde, caso a composição não conforme com a experiência não é possível concluir a demonstração (ou provar o teorema). Quando uma análise bergsonista é mal sucedida, ela produz um efeito cognitivo semelhante

ao do *halting-problem* em Máquinas de Turing. Cabe ressaltar que da mesma forma que ocorre na matemática, a verificação bergsonista não garante certeza absoluta mas apenas certeza suficiente.

3.3 As Dificuldades na Investigação Metafísica

Bergson distingue a ciência da metafísica pelo fato da primeira se realizar no âmbito simbólico e a segunda dispensar os símbolos [Bergson, 2006]. Em outras palavras, para Bergson a ciência centra esforços no estudo da extensão e a metafísica direciona seu foco para a duração [Bergson, 2006]. A dificuldade encontrada no estudo da metafísica consiste justamente em abordá-la por meio do ferramental simbólico [Bergson, 2006].

A metafísica de Bergson toma a vivência como objeto de estudo. Por isso, diferente da metafísica aristotélica que se baseia nas quatro causas do ser, a metafísica bergsonista entende que a essência é a duração. Ao considerar a essência como duração, por um lado ela passa a se apresentar como multiplicidades de nuanças e por outro lado ela é colocada no fluxo e se apresenta como mudança [Bergson, 2006]. A vivência é portanto algo holístico, homogêneo e em constante mudança [Bergson, 2006]. Este é o objeto do bergsonismo.

A descrição de uma vivência, por usar símbolos, faz com que ela deixe de ser holística e passe a ser parcial. Também faz com que ela deixe de ser homogênea e passe a ser fragementada. Além disso, faz com que ela perca seu dinamismo ou *devir* e passe a ser estática. Esses problemas são oriundos das propriedades inerentes aos símbolos e por isso não podem ser evitadas.

O que Bergson critica na metafísica não é a descrição mas o processo de semiose que é feito sobre ela [Bergson, 2006]. Grosso modo, a semiose é um processo onde se toma uma interpretação como objeto de reflexão e sobre ela cria-se um nova interpretação [Peirce, 2012]. Na visão de Bergson, a aplicação sucessiva deste processo faz com que as interpretações se tornem cada vez mais abstratas. Segundo Teixeira em [Teixeira, 2001], "a abstração atribui realidade ao que é parcial, ao que não existe por si, nem se explica por si". Nesta forma de raciocínio toma-se o conceito como objeto de estudo, ignorando o objeto em si. Ou seja, conforme as semioses se sucedem perde-se cada vez mais contato com a vivência propriamente dita.

O meio utilizado por Bergson para contornar este problema, consiste em inverter o hábito de se pensar por meio de conceitos e, num primeiro momento, tomar contato direto com o objeto, sem deixar que conceitos se interponham na vivência [Bergson, 2006]. Isso requer que, para cada investigação, se empreenda um esforço inteiramente novo na

tomada de contato real com o objeto. Feito isso, os conceitos podem ser reinseridos, desde que oriundos da experiência com o real (e não o contrário, como se costuma fazer). Esta abordagem, segundo Bergson, é o que propicia o aumento na precisão das investigações metafísicas, além de evitar diversas armadilhas oriundas do pensamento conceitual/abstrato no entendimento do real.

Em outro termos, Bergson sugere que dada uma descrição de vivência, por um esforço de intuição, deve-se buscar inserir-se na vivência. Uma vez que ela esteja sendo "vivenciada" na mente é que se pode conduzir a crítica do estudo metafísico. Percebe-se daí que o bergsonismo é intelectual, isto é, o estudo metafísico é realizado na mente de uma pessoa.

3.3.1 Framework de Investigação Bergsonista

O método de investigação bergsonista foi estabelecido em [Bergson, 2006], no entanto ele não foi apresentado de forma conveniente no sentido de poder ser fácil e amplamente replicável como ocorre com o método científico. Foi Deleuze em [Deleuze, 1999] tentou suprir esta deficiência de apresentação estabelecendo as três regras do método da intuição. De maneira geral, com exceção da forma, não existe nenhuma diferença marcante entre o método de Bergson e o de Deleuze.

- **Primeira regra:** Aplicar a prova do verdadeiro e do falso aos próprios problemas, denunciar os falsos problemas, reconciliar verdade e criação no nível dos problemas. (...) Os falsos problemas são de dois tipos: "problemas inexistentes", que assim se definem porque seus próprios termos implicam uma confusão entre o mais e o menos; "problemas mal colocados", que assim se definem porque seus termos representam mistos mal analisados [Deleuze, 1999].
- **Segunda regra:** Lutar contra a ilusão, reencontrar as verdadeiras diferenças de natureza ou as articulações do real. (...) O real não é somente o que se divide segundo articulações naturais ou diferenças de natureza, mas é também o que se reúne segundo vias que convergem para um mesmo ponto ideal ou virtual [Deleuze, 1999].
- **Terceira regra:** Colocar os problemas e resolvê-los mais em função do tempo do que do espaço [Deleuze, 1999].

Este estudo reconhece o mesmo problema que Deleuze parece ter reconhecido na obra de Bergson. No entanto, sugere que a forma apresentada por Deleuze tampouco é adequada no estudo da programação. Assim, uma nova forma mas que se considera

adequada será apresentado neste estudo. Cabe ressaltar que com exceção da forma, não existe nenhuma diferença marcante entre o método apresentado e o de Bergson e Deleuze.

Duração, Extensão e Multiplicidade

Pelo bergsonismo, uma vivência pode ser dividida em multiplicidades de extensão e de duração [Deleuze, 1999]. A duração se subdivide qualitativamente mais uma vez [Deleuze, 1999]. Isto significa que dentro da duração é possível encontrar novamente multiplicidades de extensão e duração. Apenas nesta segunda divisão é que se torna possível dispensar os símbolos por completo e ter acesso à duração em seu estado "puro" [Deleuze, 1999]. Uma representação desta estrutura é apresentada na figura 3.1.

Embora a duração deva ser subdividida, Bergson não mostra muito interesse pela divisão extensão pelo fato de que a essência, foco de seu interesse, estar sempre no lado da duração [Deleuze, 1999]. Se diz isso pelo fato de que extensão tem a propriedade de subdividir-se indefinidademente em elementos cada vez menores a ponto de assemelharem-se, geometricamente, quimicamente, *et cetera*, à qualquer outro objeto [Deleuze, 1999]. Em outras palavras, esta dificuldade em dividir a extensão é o que cria o *problema da divisão do misto*. Além disso, o preceito de que não se deve dividir a extensão, cria um nível de abstração (mantém o nível da percepção) adequado e ajuda a estabelecer um universo de discurso para vivência.

Relação Multiestável de Tendências

A relação entre os elementos do FIB deve ser multiestável. A relação multiestável ocorre quando um único objeto se apresenta ao mesmo tempo como várias coisas, suscitando assim mudanças espontâneas e estocásticas na percepção [Lehar, 2003], um exemplo de desenho multiestável pode ser visto na figura 3.2. Isto significa que ao dividir uma vivência com a intenção de conformá-la ao FIB, o resultado deve ser um conjunto de elementos sobre os quais a atenção oscila espontânea e estocasticamente.

A existência de uma relação multiestável entre um conjunto de elementos significa que, na vivência, eles não podem ser dissociados. Esta é uma propriedade fundamental para o FIB pois é o que garante a referência à um único objeto de estudos. Isso significa que, em termos de percepção, os elementos resultantes do FIB são como tendências que se sobrepõe continuamente. Caso os elementos pudessem ser dissociados não seriam mais tendências qualitativas do objeto de estudos mas elementos independentes numa relação de composição quantitativa.

Figura 3.2: Exemplo de figura multiestável [Lehar, 2003]

Como consequência da multiplicidade que cada um dos elementos representa, cabe considerar ainda que, embora cada elemento do FIB deva ser nomeado, estes nomes devem expressar a multiplicidade que representam. No entanto, não se deve buscar por nomes que abstraiam a diferença entre os elementos mas por uma que os suscite [Deleuze, 1999]. Considere, como exemplo, que se deva considerar o vermelho e o azul como nuanças de uma mesma coisa, uma opção é abstrair sua diferença e reuní-los sob o gênero "cor"; outra é apresentar tais objetos com a palavra multiestável "luz branca" a qual, de maneira multiestável, representa estes e outros elementos ao mesmo tempo [Deleuze, 1999].

Memória como Artefato

As sucessivas interposições das tendências de uma duração se consolidam na forma de memória. Esta memória pode ser uma construção mental ou artefato. Em outros termos, a memória é o produto da vivência. No entanto, isso não trata de acompanhar a memória interiorizada mas a memória objetiva na forma de um artefato seja ele uma construção mental ou um produto físico. Em outros termos, trata de acompanhar a memória da extensão pois a memória da duração é inexprimível. O acompanhamento da memória é relevante no sentido de ajudar a separar e organizar aquilo que é vivência e aquilo que resulta da vivência.

Mecanismos de Precisão

A investigação metafísica, tal qual a investigação matemática *cf.* [Brouwer, 2011], ocorre sobre construções mentais. No entanto, a matemática é considerada uma ciência exata enquanto a metafísica não. Bergson concorda com esta afirmação, denunciando que falta precisão à filosofia [Bergson, 2006]. Embora Bergson também seja acusado de faltar com a precisão [Russel, 1912] e [], possivelmente tal acusação se deve por um lado à complexidade do bergsonismo e por outro à uma questão de forma.

A complexidade do bergsonismo reside no fato dele subverter o hábito do pensamento conceitual. Como num primeiro momento ele ignora toda a extensão na busca por um contato direto com a vivência, o ponto de partida de suas idéias é diferente. Um avaliador que não esteja ciente disso não consegue acompanhar a gênese de sua argumentação. Quando posteriormente os conceitos são reencontrados para recompor o misto, o avaliador não compreende porque eles foram retomados uma vez que foram denunciados. Em suma, a complexidade do bergsonismo se dá ao tentar entender algo que se propõe a descrever o absoluto e o dinâmico sob a ótica de algo parcial e estático. Isso fica especialmente evidente na crítica de Russell em [Russel, 1912].

Com relação à forma, é fato que a descrição filosófica difere da prova de um teorema. Grosso modo, a descrição de uma teoria metafísica se dá por construção e a prova de um teorema se dá por dedução [Deleuze & Guattari, 2010]. Enquanto na filosofia se elabora uma idéia até chegar na conclusão; na matemática se apresenta uma conclusão e se tenta então demonstrá-la de forma suscinta. Neste sentido é possível afirmar que o método matemático é mais objetivo que o filosófico pois impede divagações.

Outro ponto que merece atenção tem relação à forma com que a idéia é elaborada. Tanto nos sistemas filosóficos como nos sistema axiomáticos, é possível lançar mão das regras conforme a conveniência. A diferença é que nos sistemas axiomáticos, as regras costumam ser explicitadas e a cada dedução informa-se qual regra foi utilizada. Nos sistemas filosóficos as regras são apresentadas de forma implícita conforme a idéia é desenvolvida e utilizadas também de maneira não declarada.

Embora sejam apenas questões de forma, ela influem em como as pessoas entendem as teorias [Kuhn, 2005]. Na Ciência da Programação, por ser considerada uma área de exatas, a forma filosófica será considerada estranha o que pode fazer com que as idéias apresentadas na forma de filosofia sejam ignoradas [Kuhn, 2005]. Neste sentido, embora o FIB seja um *framework* filosófico ele assumirá a forma matemática onde cada divisão proposta receberá o nome de "enunciado" o qual será seguido por uma "verificação de adequação" da consistência da divisão.

Das seções antecedentes, é possível derivar as regras: Identificação e caracterização da extensão (origina forma/dimensão), extensão da duração (origina símbolos/conceitos), duração (origina vislumbres) e suas multiplicidades; Avaliação da validade da divisão (busca por "divisões inexistentes" e "divisões mal-colocadas"); Reencontro com o real por meio de intuição e identificação da relação multiestável entre as tendências; Constatação de convergência entre duração e extensão ao reconstruir o misto; Validação da multiestabilidade dos termos na produção de nuanças; e Identificação do produto da vivência (memória). Grosso modo, esta é uma outra organização para as regras propostas por Deleuze em [Deleuze, 1999].

Se bem montado, o FIB de uma pessoa deve coincidir com o FIB de uma outra pessoa, ou melhor, outras pessoas devem reconhecer no enunciado proposto sua vivência pessoal. Não é o termo usado para nomear os elementos do FIB que devem coincidir mas a postura mental que eles procuram expressar. Caso a sensação coincida, se diz que o enunciado é válido, caso contrário é possível criticá-lo, refiná-lo ou descartá-lo. Naturalmente a validação é *à postériori* e cabe ao leitor, tal qual ocorre na ciência, realizá-la.

3.3.2 Redução do Framework de Investigação Bergsonista

O *Framework* de Divisão Bergsonista (FIB) apresentado na figura 3.1 apresenta dois níveis de investigação do objeto. No primeiro nível divide-se a vivência em forma e intenção, depois divide-se a intenção em idéia e preocupação. Quando se consegue a segunda divisão a duração do primeiro nível pode ser eliminada por ela ser utilitária, servindo como apoio durante a investigação.

Assim, o FIB pode ser reduzido em extensão composto por forma/dimensão, extensão da duração composto por símbolos/conceitos e duração que é a vivência em si, sem intermediários. Em outras palavras, durante a investigação de uma vivência primeiro se deve identificar e desprezar a forma, depois se deve identificar e desprezar os símbolos/conceitos para então ser possível supor a duração da vivência. Um diagrama desta redução é apresentado na figura 3.3.

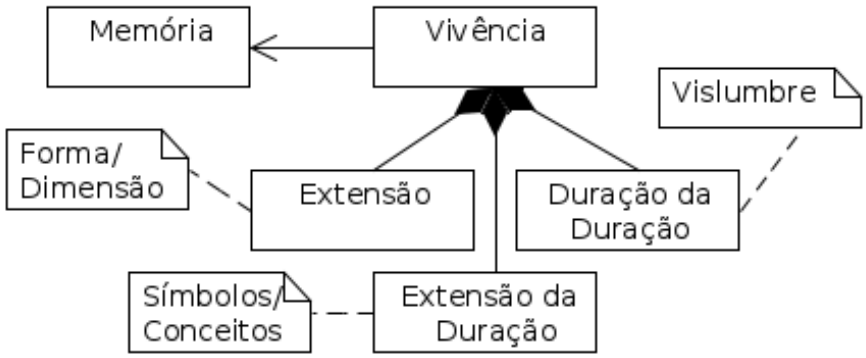

Figura 3.3: Diagrama UML do *framework* de divisão bergsonista reduzido

3.4 O Problema da Divisão do Misto

O *problema da divisão do misto* costuma ser resolvido com a definição de um critério arbitrário que seja conveniente no momento. O porém desta solução é a criação de infi-

nitas possibilidades de divisão para um mesmo objeto de estudo [Deleuze, 1999]. Isso, por sua vez, suscita uma infinidade de compreensões possíveis para uma mesma coisa. No entanto esta infinidade de compreensões, embora diferentes, são todas parecidas e como não se consegue apontar uma diferença efetiva entre elas, o conhecimento sobre o objeto se torna confuso.

Esta é uma situação bastante comum na programação. Uma vez que a programação é algo intangível, ela pode ser dividida e organizada da forma que se desejar. Isso leva à criação de uma infinidade de métodos, técnicas, ferramentas de programação que, embora sejam diferentes, são muito parecidos. Este inclusive é um dos problemas específicos apontados pelo SEMAT: "*The [existence of] huge number of methods and method variants, with differences little understood and artificially magnified.*" [Jacobson *et al.*, 2009].

Bergson observa que esta infinidade de divisões possíveis são na verdade perspectivas pela qual se estuda o objeto [Deleuze, 1999]. Ele então mostra que elas se reúnem na forma de uma multiplicidade de nuanças. Um exemplo disso são as nuanças da luz. Considere por exemplo luz nas nuanças entre o verde e o azul. Apenas no limite que esta diferença é evidente, na transição ela fica cada vez mais confusa. No entanto, também não existe diferença real entre o verde e o azul a não ser o comprimento de sua onda, assim, onde acaba o verde e começa o azul é algo arbitrário.

Esta é uma das dificuldades de se buscar uma divisão baseada na extensão [Deleuze, 1999]. Deleuze entende esse problema como uma confusão entre "mais" e "menos", ou seja, uma confusão de se querer diferenciar as coisas tomando a extensão como critério. Ele intitula este problema como "problema inexistente" por entender que não existe divisão efetiva relacionada às nuanças [Deleuze, 1999] (neste estudo o termo "divisão inexistente" poderá ser usado como sinônimo).

Outra dificuldade que ocorre quando se busca uma divisão baseada na extensão, é a mistura de elementos com diferentes naturezas [Deleuze, 1999]. Como o critério é arbitrário, pode-se incluir ou excluir qualquer coisa que se queira. Deleuze intitula este problema como "problema mal-colocado" por entender que as nuanças de uma coisa foi misturada às nuanças da outra (neste estudo o termo "divisão mal-colocada" poderá ser usado como sinônimo).

Por motivos como estes que Bergson sugere que a divisão do misto seja realizada com base na duração [Bergson, 2006]. A duração é definida como o tempo em que uma preocupação se mantém na mente e extensão como tudo aquilo que pode ser expresso por meio de símbolos ou medidas [Bergson, 2006]. Ao se remover os símbolos é possível retomar a duração, e com isso, dividir o misto conforme os movimentos da consciência. Com este tipo de análise se torna possível coisas como demarcar nuanças e diferenciar

uma duração da outra.

3.4.1 Hierarquia de Durações Bergsonista

O FIB é um instrumento que apoia a identificação da essência de uma duração da viv6encia mas não é adequado organizar os elementos de um misto. Para Bergson o misto é a forma com que o real se apresenta, ou seja, como uma mistura de graus e qualidades [Deleuze, 1999]. Uma vez que a extensão só oferece diferença de forma e dimensão, para dividir o misto é necessário organizá-lo em termos de durações. O FIB opera repelindo tudo aquilo que não faz parte da duração até encontrar sua essência. Isso impede que se consiga atingir uma idéia homogênea da vivência, ou seja, como as diversas durações se interpõe e se relacionam.

Para este intento é necessário identificar as durações e respectivas multiplicidade de nuanças para então compará-la às outras durações. A dificuldade que se tem neste sentido é que a relação existente entre vivência, duração, multiplicidade e nuança é ainda obscura *cf.* [Lawlor & Moulard, 2013]. Para buscar maior clareza nesta relação, o FIB será utilizado de maneira recursiva sobre a tendência de duração. Este procedimento propicia a investigação do meta-nível e dado que o bergsonismo é um método intelectual, o resultado desta análise tende ser puramente cognitivo.

Enunciado 1. *A* duração *pode ser caracterizada como uma convergência entre as tendências de* postura mental*,* preocupação *e* estado de espírito *as quais resultam em* vivência*.*

Adequação do Enunciado 1. *Dado que a duração é uma vivência [Deleuze, 1999], ela é passível de estudo pelo FIB.*

Relação entre as Tendências. *A duração apresenta três tendências:* postura mental *(extensão),* preocupação *(extensão da duração) e* estado de espírito *(duração). As tendencias são diferentes uma vez que elas não apresentam diferença de grau e nem incompatibilidade de natureza. Se diz que não há diferença de grau pois a postura mental se manifesta como uma "montagem mental" direcionada à um objetivo; a preocupação se manifesta de maneira objetiva; e o estado de espírito se manifesta como a "pura expressão do ser" em um dado momento. Se diz que não há incompatibilidade de natureza pois a mente toma molda conforme a preocupação (forma-idéia) e esta preocupação dura e varia de intensidade conforme o estado de espírito (idéia-vislumbre). Como o estado de espírito é inconsciente, ao invés de uma relação multiestável, a relação entre as tendências se manifesta como uma mistura homogênea composta por consciente e inconsciente.*

Descrição das Tendências. *A* postura mental *caracteriza extensão por manifestar uma "montagem mental", isto é, uma preparação da mente tendo em vista a realização de uma ação. Por exemplo, a mente se prepara, ou seja, se "monta" de maneira diferente quando se tem a intenção de cozinhar de quando se tem a intenção de dirigir. A* postura mental *apresenta multiplicidade de grau por se adequar às especificidades de cada preocupação em particular. A* preocupação *caracteriza extensão da duração por determinar o nível de percepção. Por exemplo, caso uma pessoa tenha a preocupação em dirigir, sua mente logo toma a postura de direção e a reocupação por sua vez precisa se especializar definindo o local para onde se quer dirigir. A preocupação apresenta multiplicidade de grau pela variação do seu nível de percepção. O* estado de espirito *caracteriza duração uma vez que ele é inexprimível. Considere por exemplo o estado de espírito quando se está exposto ao Sol ao final de uma tarde fria, este estado de espírito pode ser descrito como melancólico mas esta palavra está muito longe do efetivo estado de espírito naquele momento. Se pode dizer que na falta de uma descrição melhor apresenta-se uma abstração que ao menos representa uma fração do estado de espírito. O estado de espírito apresenta multiplicidade qualitativa uma vez que cada momento do estado de espírito é único. Os termos suscitam nuanças uma vez que se atualizam conforme o momento, e.g.,* postura mental do momento*,* preocupação do momento *e* estado de espírito do momento*.*

Convergência das Tendências. *O* estado de espírito *(duração) reflete o estado interior da pessoa em um dado momento e a* preocupação *(extensão da duração) denota algo que se quer realizar. A preocupação e o estado de espírito, devido sua relação consciente/inconsciente, formam um misto homogêneo que em cada momento se manifesta como uma nuança diferente. A variação da nuança pode ser tanto quantitativa pela troca do nível de percepção da preocupação quanto qualitativa pela mudança no estado de espírito da pessoa. Por sua vez, a* postura mental *(extensão) forma nuanças que se moldam para atender às variações na preocupação e no estado de espírito. O prolongamento e consolidação dessas nuanças forma o que se conhece como* vivência *(memória).*

Uma representação gráfica do FIB da Duração (enunciado 1) pode ser visto na figura 3.4a. Segundo este FIB, a essência da duração se mostra como a postura mental frente à uma preocupação a qual oscila conforme o estado de espírito.

A postura mental é uma preparação da mente para uma situação genérica. Isto pode ser percebido por um esforço de intuição ao se colocar, por exemplo, em uma duração abstrata de direção onde se sabe que vai dirigir mas nem o que e nem para onde. A postura mental se especializa quando se estabelece um veículo e um destino como preocupação. Neste ponto, a mente em termos da extensão está apta a se lançar

à ação mas ainda lhe falta o impulso vital, isto é, o estado de espírito que especializa a extensão formando assim a vivência.

É possível entender daí que apenas a nuança existe efetivamente, ou seja, a vivência se dá em função da justaposição de nuanças. Isso pode ser percebido uma vez que, conforme o estado de espírito, é possível pintar uma série de nuanças de um mesmo quadro mas nunca o quadro "em geral". Como outro exemplo, considere uma atividade que é realizada diariamente, embora a atividade seja a mesma a vivência de cada dia é única, ou seja, a vivência de cada dia se manifesta como nuança. Ainda com base no FIB da Duração, ao desconstruir a nuança rejeitando o estado de espírito se tem uma vivência virtual apresentando-se como uma multiplicidade de preocupações. Desconsiderada a preocupação a vivência se torna abstrata cuja única propriedade é durar algum tempo.

Com base nisso é possível sugerir uma Hierarquia de Durações Bergsonista como ilustrado na figura 3.4b. Esta hierarquia se sustenta enquanto teoria bergsonista uma vez que ela pode ser reduzida às concepções de Bergson sobre os estados cerebrais, mentais e da alma em [Bergson, 1920].

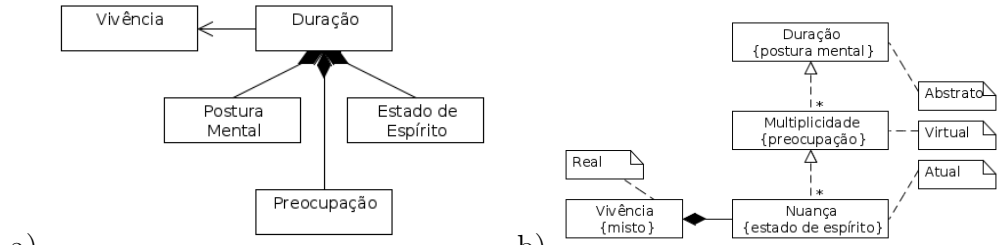

Figura 3.4: a) FIB da Duração. b) Hierarquia de Durações.

3.4.2 Framework de Comparação Bergsonista

No bergsonismo a nuança é o elemento de menor granularidade. Portanto, o maior grau de precisão que o BIF pode atingir é o dado pela nuança. A dificuldade no estudo da nuança consiste no fato desta se apresentar conforme o "estado de espírito" da pessoa no momento da vivência. Diferente do que ocorre com com a postura mental se repete como resultado de treinamento e com a preocupação que pode ser conceitualmente definida, o estado de espírito parece ter natureza inconsciente e isso parece torná-lo inacessível mesmo para a intuição. Não se consegue, por exemplo, retomar um estado de espírito da mesma forma que acontece com a duração.

Isso cria uma dificuldade na divisão do misto pois embora se consiga identificar e separar duração e multiplicidade, não se consegue fazer o mesmo com a nuança. A

opção de ignorar as nuanças também não é aceitável uma vez que impossibilitaria a divisão do misto. Cabe então a proposta de um *framework* alternativo que possibilite a identificação da Hierarquia de Durações visando a organização qualitativa do misto. Este *framework* será chamado de *Framework* de Comparação Bergsonista (FCB). Como este estudo trata de programação e ela é um processo de criação, o FCB é aplicável apenas em mistos que tratem da criação.

No bergsonismo o processo de criação de um objeto consiste em um impulso vital que transforma um vislumbre oferecido pela intuição em uma idéia e um outro impulso vital que transforma uma idéia trabalhada pela razão em algo concreto. O FCB é montado realizando o movimento inverso ao da criação de um objeto. Ou seja, dado um artefato identifica-se e elimina-se aquilo que lhe dá forma (resultado da extensão), depois identifica-se e elimina-se aquilo que lhe conceitua (resultado da extensão da duração) até que finalmente se possa supor o vislumbre que o originou (resultado da nuança). Uma representação gráfica do FCB pode ser visto na figura 3.5.

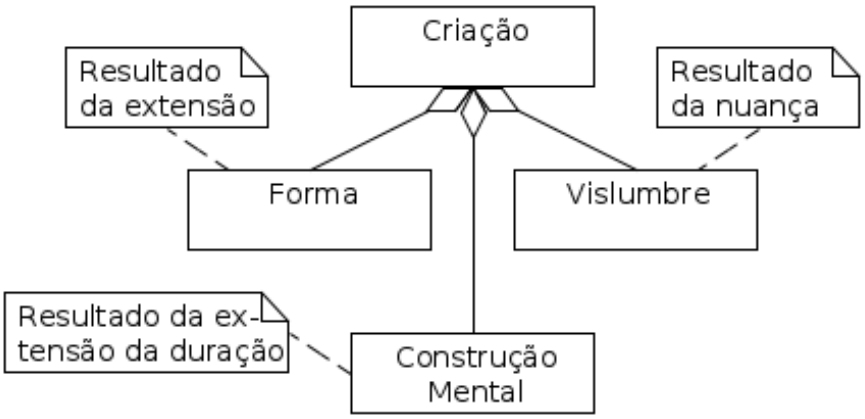

Figura 3.5: Diagrama UML do *Framework* de Comparação Bergsonista

Uma vez que duas criações apresentam um mesmo tipo de vislumbre pode-se dizer que sejam nuanças de uma mesma coisa. Se isso é verdade, o vislumbre de uma nuança deve ser passível de redução ao vislumbre da outra, isto é, deve haver uma variação de grau entre um vislumbre e outro. Quando a construção mental é compartilhada entre diferentes criações, significa que ela está fazendo o papel de memória e isso denota variação de multiplicidade. A inexistência de semelhança sugere diferença na natureza da duração.

Capítulo 4

Estudo Bergsonista da Programação

A análise metafísica da programação será conduzida aplicando os conceitos do bergsonismo sobre a vivência que se tem ao programar um computador. Se diz análise metafísica pois este estudo trabalha com a premissa de que a vivência de programação seja um objeto metafísico *cf.* capítulo 2 o qual pode ser estudado "cientificamente" por meio dos instrumentos de investigação oferecidos pelo bergsonismo *cf.* capítulo 3. Como este estudo se propõe a apresentar uma prova de conceito, ele buscará simplificar ao máximo o objeto com a intenção expor a aplicação do método em sua análise.

Precisão no bergsonismo é o quanto uma explicação se refere à especificamente uma coisa. Neste sentido, tanto maior a precisão da análise quanto maior a especificidade do objeto. Isso se deve ao fato de que conforme o tipo do objeto as preocupações envolvidas na vivência e consequentemente as durações que se interpõe variam. Quanto mais genérica for a vivência maior a possibilidade da análise desprezar durações quando ela precisar ser descrita de forma discreta e sob uma perspectiva.

Isso significa que idealmente a análise da programação deveria ser aplicada à cada desenvolvimento em particular. Este grau de especificidade é útil em casos como os de estudar ferramentas ou outros aspectos específicos da programação. No entanto, estudar a programação desta forma implicaria em estudar apenas desenvolvimentos pontuais e isso dificilmente levaria à uma compreensão geral sobre a programação. Assim, é preciso elevar o nível de percepção de forma que esta possa explorar classes de programas. Espera-se chegar em um resultado genérico mas que possa ser especializado por meio de intuição para cada problema específico dentro daquela classe de programas.

Assim, a análise bergsonista será aplicada ao desenvolvimento de softwares acadêmicos. Os softwares acadêmicos *cf.* [Dijkstra, 1969], são geralmente programas com até 500 linhas de código onde não há preocupação real com requisitos, qualidade, processo,

equipe, implantação, *et cetera*. São programas *ad hoc* com curtíssimo ciclo de desenvolvimento e expectativa de vida. Ou seja, são programas que devem caber no espaço de uma aula [Dijkstra, 1969]. A compreensão sobre este tipo de desenvolvimento se justifica no sentido de apoiar as iniciativas voltadas ao ensino da programação.

4.1 O Problema da Extensão na Programação

Antes de proceder com a investigação bergsonista da programação, convém verificar se no âmbito da programação a afirmação de Bergson de que em termos de forma ou dimensão um objeto se assemelha a qualquer outro que se queira, se sustenta.

Definição 1. ***Linguagem U****: São elementos da Linguagem U,* $\Sigma = a_i, b_n$ *onde a unicidade de cada elemento é definida por seu índice, e.g.,* a_x *e* a_x *são um mesmo elemento enquanto* a_x *e* a_y *são elementos diferentes. Uma termo da Linguagem U é definido pela expressão regular* $p = a_i b_n a_i$*, a cadeia da Linguagem U é definida por uma recorrência sobre p, i.e.,* $w = p^+$*, e.g.,* $w = (a_i b_n a_i), (a_i b_n a_i), \ldots, (a_i b_n a_i)$*. A atribuição dos índices é livre, por isso, uma cadeia pode conter tanto sub-cadeias encadeadas como não encadeadas, e.g.,* $w = (a_x b_1 a_y)(a_y b_2 a_z)$ *é uma cadeia encadeada e* $w' = (a_s b_1 a_r)(a_y b_2 a_z)$ *é uma sub-cadeia não encadeada. Concatenando w e w' se obtém a cadeia* $w \cdot w' = (a_x b_1 a_y)(a_y b_2 a_z)(a_s b_1 a_r) \in U$*. A inclusão e remoção de termos em w é arbitrário.*

Teorema 1. *Dado um conjunto de decisões arbitrárias é possível transformar qualquer cadeia da Linguagem U em outra cadeia da Lingaguem U.*

Prova 1. *Ao interagir com a Linguagem U as operações possíveis são modificar, incluir e remover termos. Por isso serão apresentadas provas para cada um desses casos.*

(1a) Substituição de Elementos em Termos Base: Sejam os termos $p = a_x b_1 a_y$ *e a* $p' = a_x b_1 a_z$*, p se torna p' ao substituir o elemento 'a*$_y$*' em p pelo elemento 'a*$_z$*'. Indução: Dado um termo* $p = a_i b_n a_i$ *e outro termo p' qualquer, p se torna p' ao substituir os elementos de p pelos de p'.*

(1b) Substituição de Termos em Cadeias Base: Sejam as cadeias $w = (a_x b_1 a_y)(a_y b_2 a_z)$ *e* $w' = (a_x b_1 a_k)(a_k b_2 a_z)$*, por (1a), w se torna w' ao substituir todas as ocorrências do elemento 'a*$_y$*' em w pelo elemento 'a*$_k$*'. Indução: Dada uma cadeia* $w = (a_i b_n a_i)(a_i b_n a_i)$ *e outra cadeia w' qualquer, de igual tamanho, por (1), w se torna w' ao substituir em w os elementos de p pelos p'.*

(2) Inclusão de Termos em Cadeias Base: Sejam as cadeias $w = (a_x b_1 a_y)$ *e* $w' = (a_x b_1 a_y)(a_y b_2 a_z)$*, w se torna w' ao inserir o termo '*$(a_y b_2 a_z)$*' em w. Indução: Dada*

uma cadeia $w = (a_i b_n a_i)$ e outra cadeia w' qualquer, de tamanho maior que w e com $w \in w'$, w se torna w' ao incluir em w os termos de w'. Observação: Caso $w \notin w'$ aplique (1b) em alguma sub-cadeia de w.

(3) Exclusão de Termos em Cadeias Base: Sejam as cadeias $w = (a_x b_1 a_y)(a_y b_2 a_z)$ e $w' = (a_x b_1 a_y)$, w se torna w' ao remover o termo '$(a_y b_2 a_z)$' de w. Indução: Dada uma cadeia $w = (a_i b_n a_i)(a_i b_n a_i)$ e outra cadeia w' qualquer, de tamanho menor que w e com $w \in w'$, w se torna w' ao remover de w os termos que não estão presentes em w'. Observação: Caso $w \notin w'$ aplique (1b) em alguma sub-cadeia de w.

Corolário 1: Seja ai uma classe do diagrama de classes UML e b_n uma associação do diagrama de classes UML. Em *termos estruturais*, dado um conjunto de decisões arbitrárias é possível transformar qualquer diagrama de classes UML em um outro diagrama de classes UML. Observações: O MOF carateriza um diagrama UML pela existência de no mínimo 2 classes e uma associação [Miller *et al.*, 2010]. O MOF define a estrutura de uma classe em termos de um diagrama de classes UML [Miller *et al.*, 2010], assim, por recorrência, este corolário é válido também para os métodos e atributos da classe. A linguagem U pode ser estendida para conformar com o diagrama de classes UML incorporando tipos de elementos 'a' e 'b', por exemplo, 'a^1' pode representar classes, 'a^2' interfaces e assim por diante.

Corolário 2: Seja ai um objeto de um diagrama de Objetos UML, bn um link de um diagrama de Objetos UML. Em termos de run-time, dado um conjunto de decisões arbitrárias é possível transformar qualquer diagrama de objetos UML em outro diagrama de objetos UML.

Teorema 2. *Dado um conjunto de decisões arbitrárias é possível transformar qualquer Máquina de Turing em outra Máquina de Turing desde que ambas possuam o mesmo alfabeto. Observação: A rigor, este teorema se aplica apenas à Máquinas de Turing com mesmo alfabeto e conjunto de estados, no entanto, como o conjunto de estados pode não afetar no resultado final da computação optou-se por exigir apenas a equivalência no alfabeto desde que se pudesse inserir ou suprimir estados arbitrariamente.*

Prova 2. *Ao conceber Máquinas de Turing as decisões possíveis são modificar, incluir e remover funções de transição. Por isso serão apresentadas provas para cada um desses casos.*

(1) Substituição de Termos em Função de Transição Base: Sejam as funções de transição $f = (q_0, a) \rightarrow (q_1, b, D)$ e $f' = (q_0, a) \rightarrow (q_1, a, D)$, f se torna f' ao substituir o termo 'b' em em f por 'a'. Indução: Dada uma função de transição $f = (Q_i, \Sigma_j) \rightarrow (Q_i, \Sigma_j, D)$ e outra função de transição f' qualquer, f se torna f' ao substituir os termos de f pelos de f'.

(2) Inclusão de Funções de Transição em Máquina de Turing Base: Sejam as Máquinas de Turing $M : \delta = (q_0, a) \rightarrow (q_1, b, D)$ *e* $K : \delta = (q_0, a) \rightarrow (q_1, b, D)$, $(q_1, b) \rightarrow (q_0, a, E)$*, M se torna K ao inserir a função de transição '*$(q_1, b) \rightarrow (q_0, a, E)$*' em M. Indução: Dada uma Máquina de Turing* $M : \delta = (Q_i, \Sigma_j) \rightarrow (Q_i, \Sigma_j, D)$ *e outra Máquina de Turing K qualquer, de tamanho maior que M e com* $M \in K$*, M se torna K ao incluir em M as funções de transição de K. Observação: Caso* $M \notin K$ *aplique (1) em alguma função de transição de M.*

(3) Exclusão de Funções de Transição em Máquina de Turing Base: Sejam as Máquinas de Turing $M : \delta = (q_0, a) \rightarrow (q_1, b, D), (q_1, b) \rightarrow (q0, a, E)$ *e* $K : \delta = (q_0, a) \rightarrow (q_1, b, D)$*, M se torna K ao remover a função de transição '*$(q_1, b) \rightarrow (q_0, a, E)$*' em M. Indução: Dada uma Máquina de Turing* $M : \delta = (Q_i, \Sigma_j) \rightarrow (Q_i, \Sigma_j, D)$ *e outra Máquina de Turing K qualquer, de tamanho menor que M e com* $M \in K$*, M se torna K ao remover em M as funções de transição que não estão em K. Observação: Caso* $M \notin K$ *aplique (1) em alguma função de transição de M.*

Corolário 3: Dado que toda Máquina de Harel é reduzível à uma Máquina de Turing numa relação similar à que se tem entre linguagens de terceira geração e linguagens de máquina, e que o comportamento de uma classe é definido em termos de uma Máquina de Harel; é possível inferir que, em *termos comportamentais*, dado um conjunto de decisões arbitrárias é possível transformar qualquer statechart UML em um outro statechart UML.

Corolário 4: Dado que o diagrama de colaboração de objetos UML é a representação de um particular cenário de uso de um statechart UML e dado que é possível, por meio do Corolário 3, transformar qualquer statechart UML em um outro statechart UML. É possível afirmar que termos de run-time, dado um conjunto de eventos arbitrários e um mesmo statechart, é possível transformar qualquer diagrama de colaboração de objetos UML em um outro diagrama de colaboração de objetos UML.

Corolário 5: Considere que dado um conjunto suficiente de decisões de projeto é possível transformar: por meio do Corolário 1, um diagrama estrutural estático em outro do mesmo tipo; por meio do Corolário 2, um diagrama estrutural dinâmico em outro do mesmo tipo; por meio do Corolário 3, um diagrama comportamental estático em outro do mesmo tipo; e por meio do Corolário 4, um diagrama comportamental dinâmico em outro do mesmo tipo. Afirma-se daí que dado um conjunto suficiente de decisões de projeto é possível transformar um modelo em qualquer outro.

4.2 Caracterização da Programação de Computadores

Caracterizar significa identificar as características de um objeto de forma que ele possa ser diferenciado de outros. Quando não se consegue fazer isso as fronteiras do objeto se tornam nebulosas e o objeto se confunde com outros elementos do misto. Portanto, a caracterização adequada de um objeto de pesquisa é o que possibilita que ele seja reconhecido como tal e então estudado. Para isso, o FCB será utilizado como ferramenta.

Durante o Desenvolvimento de Software Acadêmico, como acontece em qualquer atividade, diferentes durações e nuanças de durações se interpõe para formar o misto da vivência. Assim, para que a programação possa ser estudada é preciso caracterizar sua duração separando-a das outras que compõe o misto, como por exemplo a duração de análise, e identificar suas nuanças, como por exemplo, a nuança de modelagem.

O processo para caracterizar a programação consiste em se colocar na vivência de Desenvolvimento de Software e por meio de um esforço de intuição "observar" as preocupações que se interpõe ao longo do tempo. Ao se colocar na vivência de Desenvolvimento de Software Acadêmico, percebe-se a interposição, ao menos das durações de Análise, *Design*, Implementação e Testes (ADIT).

Mesmo no caso de softwares acadêmicos, é possível perceber que a mente oscila espotâneamente para a Análise sempre que não se sabe qual funcionamento a máquina deve apresentar. Do mesmo modo, sempre que se escreveu um trecho de código, a mente oscila espotâneamente para o Teste no sentido de verificar se aquilo que foi escrito produz o efeito que se espera. Além disso, quando se vai escrever um código a mente oscila espotâneamente para o *Design* visando realizar uma exploração ou planejamento da implementação. Estando em qualquer duração, a mente também oscila espotâneamente para a implementação sempre que um trecho de código pôde ser vislumbrado. Se diz portanto que são durações uma vez que se interpõe de maneira multiestável sem haver necessidade de sua formalização para que aconteçam.

Uma vez que as principais durações dos softwares acadêmicos foram identificadas, novamente por um esforço de intuição é necessário identificar a intenção essencial de cada tendência realizada como artefato. Isso pode ser identificado considerando como a duração se manifesta em cada tendência, ou seja, como ela se manifesta em termos de forma (extensão), em termos de construção mental (extensão da duração) e em termos da preocupação propriamente dita (duração). Com base nesse levantamento se torna possível comparar qualitativamente as durações identificando quais são independentes e quais formam nuanças.

A duração de Análise, em termos de forma consiste em uma descrição das funcionalidades do programa (extensão). Cada funcionalidade é derivado de um entendimento geral sobre o funcionamento que se espera para o programa (extensão da duração). Este entendimento geral é formado a partir de uma demanda do professor (duração). Assim, o FCB de Análise pode ser representado como {lista de funcionalidades $\oplus$ funcionamento esperado *oplus* demanda a ser atendida}.

A duração de *Design*, em termos de forma consiste em um conjunto de diagramas (extensão). Cada diagrama é derivado de um modelo mental que descreve o funcionamento que se espera para o programa (extensão da duração). O modelo é idealizado visando a produção de um conjunto de efeitos computacionais de interesse (duração). Assim, o FCB de *Design* pode ser representado como {conjunto de diagramas $\oplus$ funcionamento esperado *oplus* efeito computacional de interesse}.

A duração de Implementação, em termos de forma se manifesta como um conjunto de instruções (extensão). Cada instrução é derivada de um modelo mental que descreve o funcionamento que se espera para o programa (extensão da duração). O modelo é idealizado visando a produção de um conjunto de efeitos computacionais de interesse (duração). Assim, o FCB de *Design* pode ser representado como {conjunto de instruções $\oplus$ funcionamento esperado *oplus* efeito computacional de interesse}.

A duração de Testes, em termos de forma se manifesta como um conjunto de experimentos (extensão). Cada experimento é realizado em função uma espectativa de funcionamento (extensão da duração). A espectativa de funcionamento busca verificar se há conformidade entre aquilo que se fez e aquilo que se desejava fazer. Assim, o FCB de Testes tem a forma de: {conjunto de experimentos $\oplus$ funcionamento esperado $\oplus$ análise de conformidade}.

Uma representação gráfica do resultado desta análise é apresentada na figura 4.1. Convém ressaltar que os termos foram escolhidos de maneira conveniente, no entanto, a validação deste resultado não deve ater-se aos termos e sim ao aspecto da vivência que ele procura representar. Ou seja, o leitor, por um esforço de intuição deve retomar cada uma dessas durações e verificar como as tendências se diferenciam.

Pelo FCB, quando duas durações compartilham a extensão da duração, significa que ela atua como memória que se prolonga de uma duração em outra. Isso implica que, em um programa acadêmico, o "funcionamento esperado" é o elemento aglutinador das durações. Em outros termos, a consolidação das diferentes durações se dá na forma do constructo mental e é este constructo mental que propicia a interposição das durações.

Também pelo FCB, quando duas durações compartilham uma mesma intenção essencial na tendendência de duração, significa que ambas são nuanças de uma mesma multiplicidade. Se isso é verdade deve haver uma diferença de grau entre a extensão do

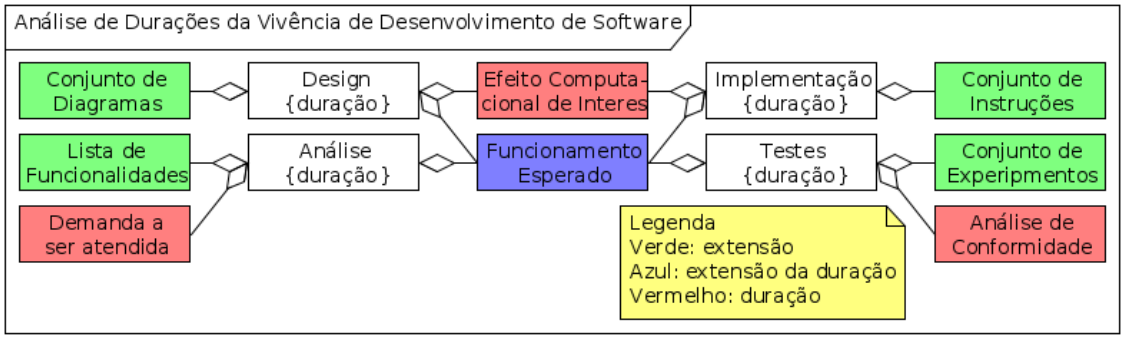

Figura 4.1: Representação UML do FCB do ADIT.

Design e da Implementação, isto é, que um diagrama pode ser reduzido à um conjunto de instruções. Isso pode ser percebido tanto pela existência de ferramentas CASE (*Computer-aided Software Engineering*) que realizam justamente este tipo de redução como por considerar que o modelo é um conjunto sucessivo de abstrações que se faz na forma de exploração, até que se consiga vislumbrar o código fonte [Martin, 2003]. Pode-se então considerar que o texto formado pelo conjunto de instruções do programa, seja também um diagrama mas na forma de um texto. Portanto, as nuanças *Design* e Implementação podem ser reunidas sob a duração de Programação.

Assim, a vivência do Desenvolvimento de Software é composta por três multiplicidades distintas: Análise, Programação e Teste. O FCB de programação para programas acadêmicos tem a forma de: {conjunto de diagramas $\oplus$ funcionamento esperado *oplus* efeito computacional de interesse}. Além disso, a programação é o meio pelo qual se espera produzir um efeito computacional de interesse.

4.3 Essência da Programação de Computadores

Uma vez que a Programação de Software Acadêmico foi recortada da vivência de Desenvolvimento de Software Acadêmico, é possível proceder com a análise de sua essência por meio do FIB. No FIB, o relato do resultado de uma investigação se dá pelo *enunciando* do resultado obtido e subsequente *verificação de adequação* do resultado às regras do FIB. Cabe ressaltar que o FIB não é um método dedutivo no sentido de que o enunciado possa ser encontrado tomando como base um conjunto de premissas e aplicação de regras de inferência.

No FIB, o *enunciado* é fruto de um esforço de intuição onde se busca encontrar os elementos necessários tomando contato com a vivência em sí. A *verificação de adequação* é um esforço da razão no sentido de verificar o resultado que foi apresentado pela intuição.

Enunciado 2. *A* programação *pode ser caracterizada como uma convergência entre as tendências de* descrição, elaboração *e* concepção *as quais resultam em* programa.

Observação 1. *Cada uma dessas tendencias é uma multiplicidade de nuanças que se interpõe conforme a preocupação em cada momento. Considere como exemplo que o programador esteja preocupado com o comportamento. Neste caso, o FIB assume uma* nuança de comportamento *onde se tem as tendências de* concepção de comportamento, elaboração de comportamento *e* descrição de comportamento *as quais resultam em* fluxo de controle. *O mesmo ocorre para outras preocupações e para diferentes níveis de granularidade.*

Observação 2. *A tentativa de colocar uma vivência em palavras só pode ser feita ao colocá-la em perspectiva, isso, por sua vez, faz com que a descrição se torne necessariamente incompleta [Bergson, 2006]. Neste sentido, ao observar os termos usados para nomear as tendências se poderia concluir que este FIB se aplica a qualquer sistema projetivo ferindo assim a idéia de precisão no bergsonismo. Embora de fato os termos possam ser compartilhados, a vivência não é. Ou seja, o arranjo cognitivo necessário para operacionalizar a programação é completamente distinto do arranjo necessário para operacionalizar uma pintura a óleo. Assim, ao ler o FIB de Programação deve-se ter em mente a vivência de* concepção durante a programação, *a vivência de* elaboração durante a programação *e a vivência de* descrição durante a programação.

Adequação do Enunciado 2. *Considerando que a programação é uma vivência [Weinberg, 1998], logo, ela é passível de ser estuda pelo FIB.*

Relação entre as Tendências. *A programação apresenta três tendências:* descrição *(extensão),* elaboração *(extensão da duração) e* concepção *(duração). As tendencias são diferentes uma vez que elas não apresentam diferença de grau e nem incompatibilidade de natureza. Se diz que não há diferença de grau pois a postura mental durante a descrição consiste em representar aquilo que se vislumbrou segundo um vocabulário específico; postura mental durante a elaboração consiste na escolha de símbolos ou conceitos que expressem um vislumbre; e a concepção procura por vislumbres que levem à solução de wicked problems de programação. Se diz que não há incompatibilidade de natureza pois a descrição propicia a representação do constructo mental produzido durante a elaboração (forma-idéia) e a elaboração consolida os vislumbres atualizados que foram oferecidos concepção (idéia-vislumbre). Entre a descrição, elaboração e a concepção existe uma sucessão natural onde a atenção se foca espontânea e estocásticas ora em uma tendência ora em outra, sugerindo a existência de uma relação multiestável entre elas.*

Descrição das Tendências. *A* descrição *caracteriza extensão uma vez a forma de um programa é determinada pelos conceitos que fundamentam a linguagem (ter-*

mos, paradigma, etc.). Além disso, as métricas de um programa são extraídas a partir da descrição. A descrição apresenta multiplicidade de grau uma vez que, em termos de forma, qualquer descrição pode ser reduzida a outra desde que os termos sejam ajustados (para quaisquer programas X e Y, se os termos de X forem ajustados para equivaler aos termos de Y, logo, X terá sido reduzido a Y). A elaboração *caracteriza extensão da duração por buscar desenvolver o constructo mental ao atualizar os vislumbres por meio de símbolos e conceitos. A multiplicidade da elaboração consiste na infinidade de possibilidades de composições de símbolos e conceitos visando atualizar o vislumbre. A* concepção *caracteriza duração por esta consistir em um vislumbre que surge, por intuição, como consequência de um misto de memória e preocupação. A multiplicidade da concepção se dá em função da infinidade de vislumbres que a mente pode conceber a cada momento. Os termos suscitam nuanças uma vez que se atualizam conforme a preocupação, por exemplo, caso a preocupação no momento seja com o comportamento, as tendências se atualizam como* descrição de comportamento, elaboração de comportamento *e* concepção de comportamento*. O mesmo ocorre para outras preocupações.*

Convergência das Tendências nos Dois Níveis. *Em uma vivência de programação, a convergência das tendências de* descrição, elaboração *e* concepção *levam à produção de um* programa*. A convergência entre as tendências se dá quando algo nebuloso como o vislumbre de um programa se realiza por meio de conceitos e se concretiza por meio de uma descrição. A descrição do vislumbre do programa é algo que prolonga entre as durações e isso o caracteriza como memória. Dada uma nova duração, a descrição do programa é alterada para compor a memória com uma nova descrição de vislumbre. Isso se repete indefinidamente até que se considere que o programa esteja pronto. A cada iteração a descrição do programa aparenta um misto homogêneo que não pode ser dissociado [Jacobson, 1995].*

Uma representação gráfica do FIB de Programação (enunciado 2) pode ser visto na figura 4.2. Segundo este FIB, a essência da programação se mostra como a descrição de um conjunto elaborado de vislumbres com a preocupação de produzir um efeito computacional.

4.3.1 Análise da Precisão e Validade do FIB

É possível verificar a precisão e validade da essência obtida submetendo o FIB) de Programação (figura 4.2 ao FCB de Desenvolvimento de Software (figura 4.1). Isso é possível uma vez que o FCB trata de artefatos e o FIB de durações. Se diz que a essência é válida se a duração coincidir com os artefatos da duração estudada e que é

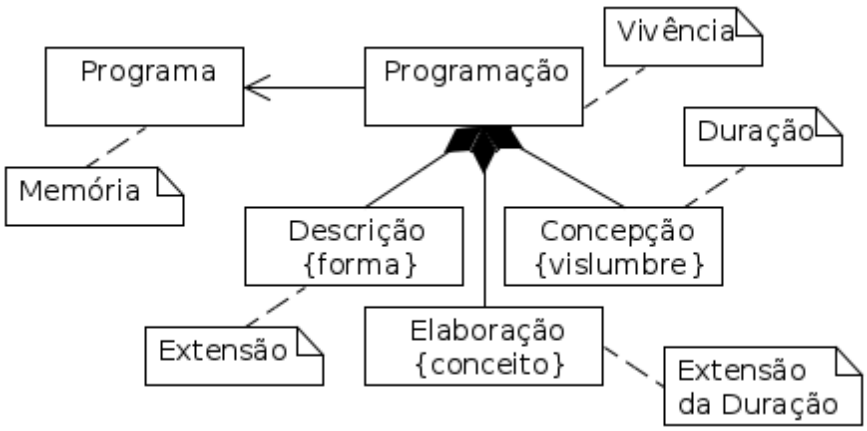

Figura 4.2: Representação UML do FIB de Programação.

precisa se ele coincidir apenas com os artefatos da duração estudada. Esta análise foi realizada por meio da listagem abaixo.

Duração de Programação *Descrição* de Diagramas (✓), *Elaboração* do Funcionamento (✓), *Concepção* do Efeito Computacional (✓)

Duração de Análise *Descrição* das funcionalidades (✓), *Elaboração* do funcionamento (✓), *Concepção* da demanda (×)

Duração de Testes *Descrição* de experimentos (✓), *Elaboração* do funcionamento (✓), *Concepção* de conformidade (×)

Perceba que o FIB é consistente com o FCB na duração de progração e por isso é válido; e inconsistente com o FCB nas durações de análise e teste e por isso é preciso. A incompatibilidade se deve ao fato de que em programas acadêmicos não faz sentido falar em concepção de demandas pois ela é dada pelo professor e nem em concepção de conformidade. Se isto for expandido para outros sistemas projetivos a inadequação aumenta ainda mais. O equivalente ao *design* na pintura à óleo seriam os esboços mas a construção mental é um efeito esperado e não um funcionamento esperado. Sugere-se que mesmo vale para outras disciplinas.

Assim, a programação consiste na: Descrição de Diagramas; Elaboração do Funcionamento; e Concepção do Efeito Computacional. Ou, de maneira suscinta, na Descrição de um Efeito Computacional.

4.4 Dinâmica da Programação de Computadores

Uma vez que a programação foi caracterizada e sua essência foi identificada, é possível investigar como a programação se dá em termos da consciência chagando assim à uma Teoria de Programação Acadêmica. Em outro termos, uma vez que a programação foi seccionada do misto e depois e dissecada na forma de tendências, é preciso recompor o misto para consolidar a compreensão sobre o objeto e consistir a compreensão alcançada junto à realidade. Uma representação gráfica desta análise pode ser vista na figura 4.3.

Para Bergson, a *Memória* é a consolidação do conjunto de vivências de uma pessoa que se prolonga entre vaárias durações. A *Duração* é um intervalo de tempo em que uma pessoa permanece em com uma preocupação em mente. Assim, a permanência em uma duração exige concentração. Quando a concentração se dispersa, diferentes durações passam a concorrer entre elas, no objetivo de se atualizarem. A mudança de duração, em geral, tem relação com preocupações relacionadas a uma tarefa. A mudança no direcionamento mental pode ocorrer de forma espontânea, quando a mente relaxa; por um estímulo da razão relacionada à algum tipo de preocupação; ou por um estímulo ligado ao problema.

Uma vez que uma Preocupação surge na mente, ela se apresenta na forma de uma tendência, de uma predisposição da mente para interpor uma determinada duração. Desde a intuição estética de [Kant, 1995] se sabe que o surgimento de preocupações exige memória. No entanto, além da memória, a decisão de interpor uma duração precisa de uma motivação que seja suficiente para justificar o esforço necessário da mente operar nessa nova duração. Cabe ressaltar que a motivação também requer memória. Sem vivências desfavoráveis (reais ou virtuais) pela não atenção à uma preocupação, o esforço necessário para tratá-la não é justificável e a preocupação acaba sendo desprezada.

Uma duração muda ou se diferencia qualitativamente. Na *Multiplicidade de Durações*, as preocupações se intercalam, no esforço de elaborar ou compreender um objeto. Mas na *Multiplicidade de Nuanças*, são as matizes de uma mesma duração que se intercalam, com o mesmo objetivo: elaborar ou compreender um objeto. Sendo o objeto em questão um programa, a Multiplicidade de Nuanças é determinada pelas durações essenciais: análise, programação e teste. Tais durações são ditas essenciais pelo fato de não ser possível criar um programa sem que as mesmas se interponham naturalmente.

Tome-se como exemplo o caso onde um programador pretende ignorar a etapa de análise, passando diretamente para a programação. Fazer isto é impossível! Uma vez dentro da duração de programação, sua preocupação se torna produzir um efeito computacional. No instante em que se começa a decidir qual é o efeito computacional

a ser produzido, ele já foi direcionado para a duração de análise. Algo semelhante ocorre com a duração de testes. Ao observar um conjunto de instruções, o programador fatalmente se preocupará em saber se o comportamento que foi descrito produz o efeito computacional desejado. Neste instante, a sua mente já foi direcionada para a duração de teste.

Há uma outra nuança de interesse, em especial, na duração de programação: a de *manutenção*. Ela consiste no processo de apreender uma nuança de programa para tranformá-la em outra, a fim de que este acompanhe as variações no mundo do problema. Esta nuança não é essencial no sentido de que ela se interponha de forma inevitável ou que se ela for desprezada inviabiliza a construção do programa. Assim, esta nuança requer motivação para que ela se interponha e se mantenha.

Algo semelhante ocorre com as durações de análise e teste. Embora sejam durações essenciais, se não houver motivação o suficiente, elas duram o mínimo necessário para que o desenvolvedor consiga retomar a duração de programação, na qual ele tem maior entusiasmo. Esse fenômeno parece ser causado por *cognitive bias cf.* [Stacy & MacMillan, 1995], mas a exploração desta particularidade na mudança de duração foi desconsiderada no escopo deste estudo.

Como uma tentativa de tratar a preocupação que provoca a duração, a mente faz uso da memória em busca de alternativas. Quando esta busca ocorre de modo consciente, é chamada de *razão*; se ocorre de modo inconsciente, *intuição*. A razão opera sobre abstrações e regras, enquanto que a intuição o faz sobre a memória (vivências acumuladas) e duração (preocupação do momento). Em outros termos, pode-se dizer que a razão opera sobre o artificial, e a intuição opera sobre o real. Desta forma, razão e intuição são complementares, no sentido de que uma corrige ou refina a outra, e interdependentes, no sentido de que uma sugere a outra.

Do bergsonismo, entende-se que *Símbolo* é o produto da razão e *Vislumbre* é o produto da intuição [Bergson, 2006]. Dado que o vislumbre é uma alternativa para a forma como lidamos com uma preocupação, pode-se dizer que a intuição produzirá somente os vislumbres pertinentes à duração do momento. Sendo o vislumbre fruto de um processo inconsciente, ele não é expresso com símbolos. Além disso, enquanto uma possibilidade, ele é caraterizado como virtual. Assim, tendo um vislumbre é preciso torná-lo atual e isso consiste em expressá-lo com símbolos.

O processo de expressar um vislumbre com signos pode ocorrer de duas formas. Se este vislumbre for algo conhecido, basta reproduzir os signos presentes na memória. Mas se não for conhecido, será necessário um esforço para escolher ou criar os signos que melhor o caracterizem. Por sua vez, a escolha dos signos para compor um vislumbre inédito é, também, realizada pela cooperação entre razão e intuição.

Enquanto a intuição pode ser considerada como um ato simples, a razão opera com diferentes mecanismos. Os mecanismos que [Peirce, 2012] atribui à razão são a *indução*, *dedução* e *abdução*. A indução e a dedução propiciam a realização de inferência. A abdução (*cf.* [Peirce, 2012]) ou a intuição sintética (*cf.* [Poncaré, 2012]) também propiciam a realização de inferências, mas estas com caráter de estimativa. A intuição sintética, por exemplo, propicia inferir sobre provas matemáticas baseadas na abordagem de indução finita. Mas sem a necessidade de construir toda a infinita sequência de números, como seria necessário em outras abordagens de prova.

Quando a duração de programação se manifesta com a nuança de manutenção, ocorre um processo inverso. Os símbolos devem ser removidos para que, por meio da intuição, seja possível supor a duração e o vislumbre que deram origem ao programa. O critério para o sucesso desta suposição é o retomada dos signos utilizados na expressão sintática do programa. Além da nuança de manutenção, este mesmo processo é utilizado quando o desenvolvedor precisa continuar programando e reserva um tempo para retomar uma "linha de raciocínio". Se optar por realizar apenas uma avaliação sintática do programa, o resultado será pontual e descontextualizado. Ou seja, será realizado por meio do que [Foote & Yoder, 1997] chama de "puxadinhos", levando esta arquitetura a uma estrutura do tipo "favela".

A suposição de durações também é utilizada para que estas sejam compartilhadas por uma equipe de desenvolvimento, dando origem ao "ponto de quebra"(*cf.* [Jacobson, 1995]) ou "alucinação compartilhada"(*cf.* [Booch, 2010]). Cabe considerar ainda que, para Bergson, esta equipe compartilhando uma mesma duração (visto que a duração é em si mudança) e evoluindo suas durações em ideias diversas, eles permanecerão com um mesmo entendimento e ainda assim surgirão criações sobre a duração. Neste aspecto, a suposição de durações é algo essencial para *programming in the large*.

Desta forma, tomando o par memória e duração, a mente passa a produzir vislumbres. E uma vez que um vislumbre é escolhido, torna-se necessário atualizá-lo. Este processo recebe o nome de *impulso-vital* e é realizado por meio da cooperação entre razão e intuição. O produto do impulso-vital é o artefato. Um artefato é a consolidação da atualização de vislumbres em diferentes durações. Isto é, uma multiplicidade de nuanças se interpõem durante a elaboração de um artefato. Além disso, sendo o artefato resultado da consolidação de atualizações diversas, ele é um tipo de memória.

Existem dois tipos de artefatos: o constructo mental e a especificação das computações. O constructo mental é um elemento cognitivo, conhecido como modelo na terminologia de [Peirce, 2012]. A especificação de computações é uma descrição parcial do elemento cognitivo, conhecido como diagrama na terminologia de [Peirce, 2012].

Considerando que são dois artefatos que coexistem em dimensões diferentes (cogni-

tiva e diagramática), a atualização de um vislumbre se dá em duas etapas. Na primeira etapa, o vislumbre é expresso com signos que o caracterizem, e é consolidado no modelo. Esta etapa pode ser imediata ou demorada, dependendo do ineditismo do vislumbre com relação à memória existente. Caso o vislumbre coincida com um elemento na memória, a passagem é imediata. Do contrário, será necessário um esforço direcionado à escolha ou criação dos signos que melhor caracterizem o vislumbre.

Quando o resultado do constructo mental consolidado pareça adequado, passa-se à segunda etapa, onde o modelo pode ser atualizado na forma de um diagrama. Esta segunda atualização é efetivada pela descrição do constructo mental, consolidado com primitivos da linguagem alvo. Esta passagem também pode ser mais ou menos demorada, variando conforme a adequação do constructo mental às limitações impostas pela linguagem alvo. Se o constructo mental se adequa imediatamente à linguagem alvo, a passagem também é imediata. Do contrário, o constructo mental deve ser adaptado para se tornar passível de realização.

A consolidação pode ocorrer de duas formas. Uma delas consiste em primeiro atualizar o artefato em uma duração ideal e depois adaptá-lo para consolidá-lo com o artefato que se está elaborando. A outra consiste em atualizá-lo diretamente na duração do artefato em elaboração para que a consolidação se dê de forma direta.

Uma vez que o artefato é um tipo de memória, ele se encontra ao "lado"da extensão, na visão de [Bergson, 2006]. Por isso, um artefato pode ser observado ou representado por perspectivas diversas, através da *multiplicidade de artefatos*. E cada perspectiva pode ser apresentada com formas ou notações diversas, formando *nuanças de artefatos*. Desta forma, ao falar em *multiplicidade de nuanças de artefatos*, entende-se que diferentes nuanças e perspectivas podem ser utilizadas no esforço de elaborar o artefato.

A multiplicidade de nuanças é dita horizontal, quando são usados diferentes artefatos para representar diferentes perspectivas de um mesmo objeto. Como ocorre, por exemplo, com os quadrantes de modelagem de [Vega, 2012-2015]. Se diz que a multiplicidade de nuanças é vertical, quando são usados diferentes artefatos para representar diferentes níveis de abstração. É o caso do uso de *high-level design*, *low-level design*, *design* de algoritmo, *et cetera*.

As multiplicidades de nuanças horizontal e vertical, suscitam outras multiplicidades de nuanças, conforme se busca elaborar tanto partes específicas do objeto, como a forma de unir essas partes. Isto significa que o refinamento de um dos múltiplos artefatos (ou nuanças de artefatos) pode impelir modificações aos outros. Por sua vez, isso pode originar novos vislumbres, que refletirão em um ou mais dos múltiplos artefatos ou nuanças de artefatos. Resumindo, é desencadeado um processo de refinamentos sucessivos. Cada refinamento é então consolidado no artefato de forma *incremental*,

tal qual uma *choice sequence*. Pode-se dizer, então, que este é o fundamento que sustenta o desenvolvimento iterativo e incremental, e que este processo é intrínseco à programação.

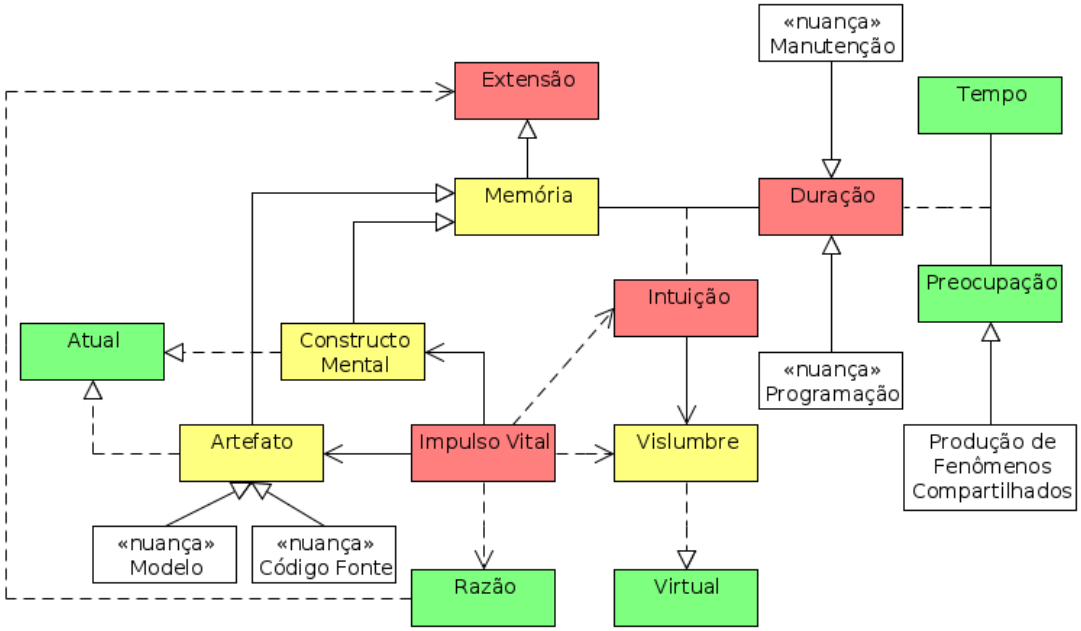

Figura 4.3: Representação UML do Misto da Programação.

* * *

A produção de um programa envolve um processo semelhante à um sistema dinâmico, onde uma multiplicidade de durações com tendências variadas concorrem para se sobrepor na produção de virtuais. Isto é, ao mesmo tempo em que se está programando, se está virtualmente analisando, testando e *vice-versa*. No entanto, quando a consciência assume uma nuança como direcionamento, e decide atualizar um virtual, enquanto ela durar, afloram apenas os virtuais pertinentes à duração em questão. Conforme o impulso-vital de uma particular atualização esmaece, novamente os virtuais de durações variadas se misturam. Cabe considerar que todas essas coisas ocorrem sempre no lado da duração. O artefato é algo inerte, que apenas reflete uma consolidação de vislumbres daquilo que ocorreu no nível da consciência.

4.4.1 Exemplo da Programação de Computadores

Uma vez que a dinâmica da programação foi esboçada, a título de exemplo, pode-se aplicá-la na construção de um Programa Acadêmico. Este exemplo ilustra a dinâmica de programação tomando como base a necessidade de um *stakeholder* saber qual a *Área*

Basal de uma árvore. Parte-se do pressuposto que o programador não tem conhecimento prévio, isto é, memória, sobre botânica e que portanto precisará desenvolvê-la. Assumiu-se esta premissa por ser uma situação comum no desenvolvimento de programas. Para facilitar o entendimento, este exemplo foi composto de forma a representar um desenvolvimento ingênuo de um programa algorítmico.

Mesmo havendo vasta memória sobre programação, a ausência de memória sobre botânica faz com que não venha à mente nenhum indício de como programar este cálculo. Então, logo surge a preocupação de saber o que é a Área Basal de uma árvore. Isto direciona a mente a uma duração de análise, onde começam a surgir vislumbres e fantasias que tentam explicar o objeto. O percurso natural da mente é tentar explorar estas imagens mentais, para tentar imaginar o que possa ser a Área Basal de uma árvore.

No entanto, esta exploração deve ser evitada quando a análise é feita com intenção de criar um programa. Isso ocorre pelo fato da maioria dos programas serem elaborados para interagir com ambientes artificiais, e por isso arbitrários. Sendo arbitrários, a memória acumulada em um ambiente não se aplica em outro, fazendo com que a memória do domínio sempre se inicie a partir do nível 1 de [Dreyfus, 1998]. Retomando o exemplo dado, é possível dizer que "Área Basal"pode ter significados diferentes conforme muda o ambiente do domínio.

Assim, define-se *cf.* [Martins, 1991], ou seja, Área Basal é área que uma árvore ocupa no solo e é defina em função da fórmula $\frac{\pi \cdot DAP^2}{4}$. Com esta informação, a duração de programação já começa concorrer para se interpor à duração de análise, pois já é possível antever vislumbres de como programar este cálculo. Este exemplo trata de um tipo de construção que já foi exaustivamente realizada pelo desenvolvedor, a atualização do vislumbre em constructo mental é imediata. Já a atualização do constructo mental em diagrama também é imediata. E embora imediata, o desenvolvedor deve decidir entre múltiplas nuanças de diagrama.

Neste ponto do exemplo, a multiplicidade de nuanças se apresenta vertical, cabendo ao desenvolvedor optar por descrever o constructo mental como pseudo-código, ou nos termos da linguagem de programação. Esta preocupação direciona a duração de programação para uma nuança de design, e logo surge o vislumbre de que se deva usar a razão. A razão, por sua vez, mostra que existem ainda elementos desconhecidos nesta fórmula, sendo melhor realizar a exploração no pseudo-código. O resultado deve ser algo parecido com “AB=(π*(DAP*DAP))/4;”.

Ao fazer isso, a mente é novamente remetida à duração de análise, buscando o significado de ‘DAP’. Novamente, segundo [Martins, 1991], DAP significa *Diâmetro à Altura do Peito* e seu valor é obtido a partir do PAP *Perímetro à Altura do Peito* pela

fórmula $\frac{PAP}{\pi}$. PAP, por sua vez, é o valor em centímetros do perímetro do tronco da árvore obtido à altura de 1,5 m. Mais uma vez, a mente se volta para a duração de programação.

Embora esta nova informação seja trivial, e passe de imediato para o diagrama, ela se consolida de forma análoga aos novos elementos de uma *choice-sequence*. O diagrama se manifesta como uma nuança da forma "DAP=PAP/π;" e "PAP $\rightarrow$ *input*;". Perceba que a mente não questionou, neste ponto, se o diagrama deveria ser ou não em pseudo-código. Isto ocorre porque os vislumbres são concebidos dentro de uma particular nuança, e as nuanças incorporam restrições. Levando em conta que a intenção seja compor um constructo mental, é preciso retomar (ou supor) a nuança do referido constructo mental. Assim, retomar uma nuança consiste em retomar as decisões e restrições que a originaram, e que estão consolidadas na memória. Desta forma, sabendo que a intuição produz vislumbres apenas no contexto memória (constructo mental) e duração (nuança), deve-se escolher o momento de tomar uma decisão ou impor uma restrição. Isto irá direcionar os vislumbres produzidos a partir de então.

Existem ao menos três nuanças que podem resultar da consolidação do cálculo do DAP com o cálculo da Área Basal. Uma delas é o resultado do encadeamento das duas contas, e por isso imediato: "DAP=PAP/π; AB=(π*(DAP*DAP))/4;". Outra possibilidade de expressar este mesmo cálculo é consolidando-o por composição: "AB=(π*((PAP/π)*(PAP/π)))/4". A terceira nuança exige algum conhecimento sobre geometria, de forma que: "AB = (PAP*PAP)/(4*π)". Da mesma forma que ocorre com a análise, esta terceira nuança deve ser evitada na programação, pois não há como garantir que a memória do programador seja aplicável ao particular domínio em que se encontra a aplicação.

Uma vez que uma das nuanças sejam atualizadas na forma de diagrama, duas durações concorrem para se atualizar: a nuança de organização e a duração de teste. Perceba que a interposição de durações ocorre em duas etapas. Primeiro surge uma preocupação, neste caso, com a organização ou funcionamento do programa. É o estágio onde a duração é potencial, cabendo ao desenvolvedor decidir se deve explorá-la ou não. Se ele decide que não há motivação o suficiente para fazê-lo, a duração é desprezada. Do contrário, ele interpõe a duração e passa a explorá-la.

A nuança de organização tem sentido de estruturar o programa, para que ele se torne passível de modificações futuras. Uma perspectiva da preocupação de organização é explorar diferentes nuanças, até que uma delas se mostre adequada. Um exemplo de duas nuanças de fluxo de controle, para o programa que calcula a Área Basal, é apresentado na figura 4.4.1. A escolha de uma nuança em detrimento da outra se dá preponderantemente pela intuição estética de [Kant, 1995]. Ou seja, embora o

vislumbre de possibilidades seja oriundo da intuição “efetiva”, a escolha da nuança é feita segundo a intuição estética, considerando o projeto e a equipe.

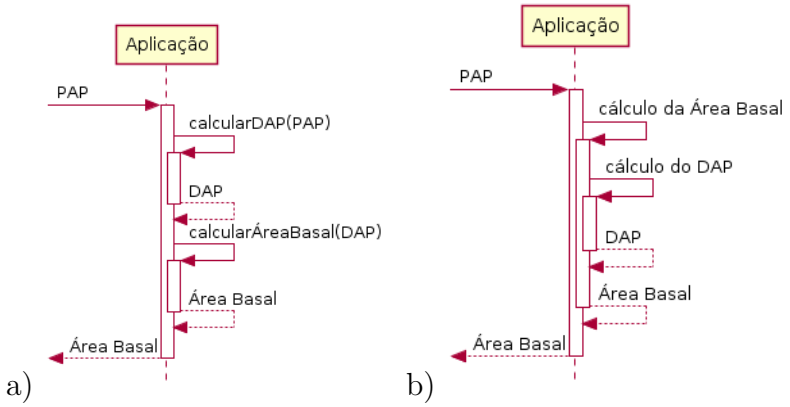

Figura 4.4: Nuanças do fluxo de controle para o aplicativo de cálculo da Área Basal

Outra perspectiva da preocupação de organização dá origem à multiplicidade de nuanças “horizontais”, como apresentada na figura 4.4.1. Cada uma dessas perspectivas suscita na mente preocupações diferentes, e isso leva à um processo de refinamentos sucessivos, para que os diagramas fiquem suficientemente consistentes. Por exemplo, a figura 4.4.1b fez com que a figura 4.4.1a fosse refinada, tornando-se a figura 4.4.1a. Naturalmente, uma infinidade de outras decisões poderiam ser tomadas, como optar por outros paradigmas, estilos arquiteturais, princípios de modelagem, *et cetera*.

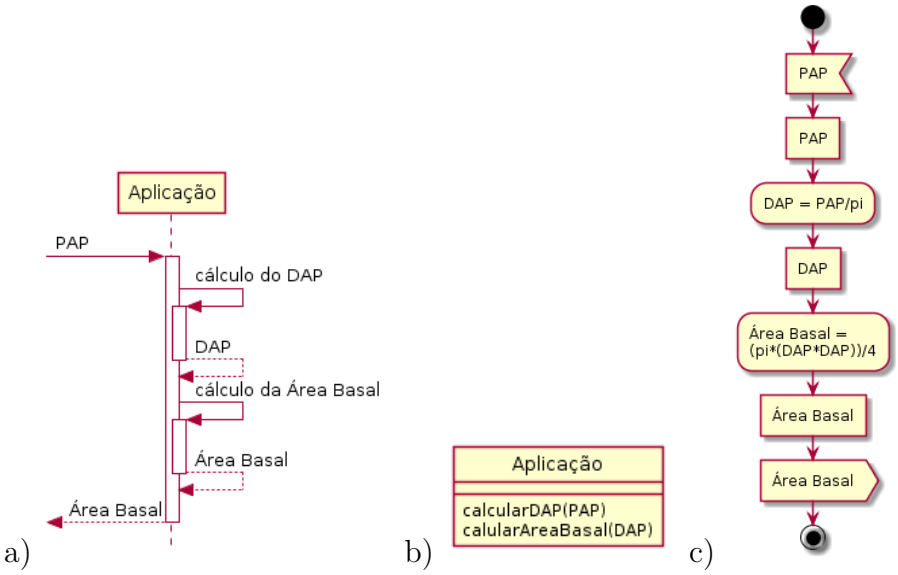

Figura 4.5: Multiplicidade de nuanças de artefatos de um mesmo

Nos diagramas da nuança de *design*, espera-se consistência suficiente, uma vez que o objetivo não é um diagrama formal, tal qual ocorre na nuança de codificação. Neste nível, pretende-se que as lacunas e inconsistências entre os diagramas sejam preenchidas, ou resolvidas por um processo de abdução. Por exemplo, a figura 4.4.1a

e figura 4.4.1c são inconsistentes. A figura 4.4.1a sugere uma organização segundo o estilo "programa-principal e sub-rotinas", enquanto a figura 4.4.1b apresenta o fluxo de controle segundo uma organização linear. No entanto, embora inconsistentes, é possível abduzir diversas maneiras em que elas sejam compreendidas de forma consistente. Este mesmo procedimento se aplica tanto na multiplicidade de nuanças "vertical", quanto na sobreposição de diagramas (diagramas diferentes, com os mesmos elementos e intenção distinta).

O mero ato de ler esses diagramas faz surgir a preocupação de saber se o resultado do cálculo será o esperado. No caso em que se decide dar atenção à esta preocupação, naturalmente a duração de teste se interpõe. Uma vez nesta duração, a razão e a intuição passam a verificar e validar a estrutura lógica das instruções: se elas produzem o efeito computacional esperado e o resultado do cálculo. Enquanto o teste está sendo realizado mentalmente, ele faz uso da intuição sintética (usada nas provas por indução finita), para se convencer da correção do programa. Decidido que as instruções realizam aquilo a que se propõe, naturalmente surge uma preocupação com a implementação, levando a mente à nuança de codificação.

A nuança de codificação suscita uma série de outras preocupações. Uma delas tem relação com a adequação do pseudo-código aos elementos da linguagem de programação. Para isso, é preciso escolher os tipos de dados, verificar a sintaxe, a semântica, e outros detalhes. Por outro lado, existem as questões tecnológicas como Entrada e Saída, etc. Dado que *design* e codificação são nuanças da programação, basta dizer que o processo cognitivo envolvido é o mesmo.

Capítulo 5

Resultados e Discussões

Em termos de resultados, este estudo derivou um conjunto de *frameworks* da obra Bergson (capítulo 3), os aplicou na análise da caracterização, essência e dinâmica da programação (capítulo 3). Convém agora derivar algumas implicações e colocar esses resultados em perspectiva com aquilo que se objetivava inicialmente (capítulo 1) e discutir se o objetivo foi ou não atingido.

5.1 Considerações sobre Programação pelo viés Bergsonista

Como foi apresentado no capítulo 4, a essência da programação pode ser expressa como: *Programa* (memória), *Descrição de Diagramas* (extensão), *Elaboração do Funcionamento* (extensão da duração) e *Concepção do Efeito Computacional* (duração). Cabe observar, com base na essência encontrada, alguns problemas reconhecidos na Ciência da Programação, para saber que tipo de conhecimento é passível de ser obtido daí. Optou-se por deixar as considerações em estágio embrionário uma vez que a proposta do estudo é uma prova de conceito, onde, ao longo do tempo as idéias trabalhadas inevitavelmente serão revisadas.

Definição de Programação. Em termos estritos a essência da programação é sua duração/extensão e o seu resultado é o programa, ou seja, *a Programação consiste na Concepção de Efeitos Computacionais Descritos na forma de um Programa.* Embora este seja uma definição bastante conhecida, ela difere de outras definições pois é oriunda de um processo racional e reflete aquilo que ocorre em termos cognitivos. Em outras palavras não se trata de uma criação da razão ou da imaginação mas uma derivação

racional oriunda de uma análise metafísica. Isso significa que diferente de outras definições esta pode ser justificada. Naturalmente as descrições que forem nuanças desta também usufruem desta propriedade.

Turing e Gödel. Naturalmente, um programa é a descrição de um conjunto de funções de transição de uma Máquina de Turing. No entanto, para Turing a função de transição representa uma mudança no estado mental do matemático enquanto este faz contas [Turing, 1950]. Gödel, no entanto, discorda dessa posição afirmando que os os estados mentais são infinitos e voláteis [Gödel, 1990]. Esta é uma outra discussão em aberto na programação. No entanto, ao considerá-la sob a luz do bergsonismo estas duas posições podem coexistir *cf.* [Bergson, 1920] desde que se considere que Turing esteja se referindo aos estados cerebrais (extensão) e que Gödel aos estados da alma (duração).

O Limite da IA. Para reforçar esta idéia, ao considerar a Máquina de Turing segundo a concepção de [Heidegger, 1977], onde a tecnologia é uma relação cuja essência consiste em "revelar" um aspecto da natureza. O moinho de vento, por exemplo, revela que uma das propriedades do vento é ser uma fonte de energia. A Máquina de Turing, por sua vez, revela que uma das propriedades da mente é funcionar como um processador mecânico de símbolos computáveis. Ou seja, a Máquina de Turing revela a propriedade de extensão na mente. Neste ponto, ocorre um fenômeno interessante. Como um estado da alma (duração) sempre se manifesta como um estado cerebral (extensão), logo, a Inteligência Artificial é capaz de simular, em princípio, qualquer ação humana que ocorra em nível cerebral. Mas isso também estabelece o limite da IA, ela não alcança a nuança.

Programa e Computação. Isso significa que a computação ocorre sempre no lado da extensão. Ou seja, a Máquina de Turing se comporta conforme for determinado por uma função de transição. Uma vez que a máquina se comporta sempre da mesma forma, em vista de uma mesma função de transição, pode-se dizer que ela se repete; ou seja, que apresenta um comportamento mecânico. Assim, é possível estabelecer uma relação de extensão entre o programa e a computação. Uma vez que o compilador também é uma máquina, é possível estabelecer uma relação de extensão entre diagrama e computação.

Paradigma Matemático. Segundo a essência da programação, o lado da extensão ocorre em duas etapas: construção mental e diagrama. Considerando que a Máquina

de Turing é um constructo matemático, o programa é a descrição de um cálculo, em termos de um conjunto de funções de transição. Então, a escrita desta descrição deve estar subordinada a um paradigma matemático. Dentre os paradigmas matemáticos descritos em [da Costa, 1992] e [Heyting, 1971], o brouwerismo é o único que apresenta esta separação em construção mental e símbolo. Talvez a forma mais direta de mostrar essa relação seja pelo isomorfismo de Curry-Howard, o qual estabelece uma relação direta entre proposições do cálculo-λ e uma prova da matemática brouwerista. A interpretação deste isomorfismo no âmbito da computação, considerando a Tese de Church-Turing (equivalência entre o cálculo-λ e a máquina automática *cf.* [Turing, 1937]), foi que existe relação direta entre um programa e uma prova da matemática intuicionista [Sørensen & Urzyczyn, 2006].

Formalismo e Brouwerismo. Embora distintos, o formalismo e brouwerismo coexistem na Máquina de Turing. O objeto do formalismo são as expressões do cálculo (ou computações) possíveis, dado um modelo meta-matemático. E o objeto do brouwerismo é concepção do modelo meta-matemático *cf.* [Heyting, 1971]. Segundo DaCosta em [da Costa, 1992] e Heyting em [Heyting, 1971], no nível meta-matemáico, por exemplo, os símbolos são dotados de semântica intuitiva e apenas são aceitas provas por construção. A existência de uma gramática implica na existência de uma linguagem, porém, na linguagem o significado meta-matemático que originou cada símbolo ou proposição é abstraído e, portanto, as cadeias desta linguagem, para [da Costa, 1992] e [Heyting, 1971], tem caráter estritamente simbólico. [da Costa, 1992] acredita que tal propriedade propicia que uma mesma linguagem formal possa ser utilizada para descrever divergentes cálculos, desde que se enquadrem em um particular sistema de regras.

Instruções Genéricas e Singulares. Com base na dinâmica de programação, que se derivou da essência no capítulo 4, é possível perceber que na atualização de um virtual existe uma variação na quantidade de esforço necessário. Embora não seja possível demarcar o limite, é possível apontar os extremos. Em um extremo, existem as instruções genéricas e no outro as instruções singulares. As instruções genéricas são aquelas destinas à solução de *tame-problems* onde passagem do virtual para a descrição se dá de maneira quase instantânea por serem virtuais reconhecidos pela memória. Um exemplo, são as instruções utilizadas para realizar a conexão com o Banco de Dados. As instruções singulares são aquelas destinadas à solução de *wicked-problems*, ou seja, aos aspectos centrais de uma particular aplicação. A atualização desses virtuais requer um processo mais sofisticado de elaboração pois não existe memória que se possa tomar como referência. Tomando como base o Princípio de Pareto (80/20), uma aplicação é

composta por 80% de instruções genéricas (destinadas à infra-estrutura da aplicação) e 20% de instruções singulares (destinadas à atender a necessidade ou meta da aplicação). Isso sugere que 20% das instruções são responsáveis por 80% das dificuldades que se encontra no desenvolvimento de software. Ou seja, poderia haver um esforço de pesquisa em programação sobre este tipo de instrução.

Crise do Software. Entende-se como crise do software a incapacidade de saber *a priori* se um artefato cumprirá seu papel no mundo do problema [Naur & Randell, 1969]. Esta incapacidade significa que o programador, muitas vezes nem por um esforço de imaginação, consegue prever o artefato no mundo do problema. Em outros termos, lhe falta memória sobre o mundo do problema que permita prever a interação do artefato com o mundo do problema por um esforço de intuição. Assim, para resolver a Crise do Software serina necessário que o programador acumulasse memória suficiente a respeito do mundo do problema para que pude prever sua interação com um artefato por meio de um esforço de intuição. No entanto cada projeto é um *wicked-problem* o que inviabiliza esta possibilidade em termos práticos (estima-se que a formação de um *expert* requeira cerca de dez mil horas - cerca de dez anos). Percebe-se assim que a Crise do Software não é algo a ser solucionado mas algo a ser administrado nos projetos de software. Isto, por sua vez, também pode ser alvo de pesquisas subsequentes.

Educação em Programação. Segundo a teoria proposta, o ensino de programação consiste, por um lado, no acúmulo de experiência necessária para a formação de memória suficiente, a qual possibilitará à intuição oferecer vislumbres. Por outro lado, o ensino deve suscitar no aluno quais preocupações devem ser consideradas para que durações de interesse possam se interpor à de programação durante a elaboração do texto do programa. A interposição de durações de interesse visando a formação de um texto computável é o que caracteriza a engenharia de software. Considere como exemplo um programa que calcule o índice de massa corporal. Caso a pessoa utilize apenas a duração de programação, uma nuança de programa será algo como 'float x, y; cin»x; cin»y; cout«x/(y*y);'. Embora este programa funcione, ele ignora uma série de preocupações referentes, por exemplo, à engenharia de software e ao fenômeno compartilhado. Em outros termos, durante a elaboração do texto deste programa nenhuma duração se interpôs à de programação, ou seja, nenhuma preocupação além da descrição de um comportamento mecânico surgiu na mente do programador. Assim, a educação em engenharia de software requer, por um lado, que o aluno fique sensibilizado à um conjunto de preocupações e, por outro lado, que ele acumule memória o suficiente para conseguir tratar cada preocupação. A determinação de um conjunto de preocupações necessárias à engenharia de software está fora do escopo deste estudo.

Poderia-se ficar interpretando diversos outros aspectos da programação com base no viés bergsonista. No entanto, os apresentados parecem ser suficientes enquanto prova de conceito.

5.2 Considerações sobre o Bergsonismo no Estudo da Programação

Cabe agora tentar apresentar uma resposta ao questionamento inicial da pesquisa: *o bergsonismo pode ser utilizado como método na Ciência da Programação (ciência em sentido lato)?*. Como apresentado no capítulo 2, considerando que exista uma ciência, uma epistemologia e uma metafísica da programação e que o bergsonismo é utilizado para estudar a metafísica da programação, a resposta é sim.

O bergsonismo é um método de investigação metafísico. Neste estudo, ele foi aplicado para estudar a programação e chegar em resultados válidos pelos parâmetros bergsonistas. Esses resultados foram úteis e possibilitaram apresentar ao menos uma resposta justificada para alguns problemas reconhecidamente intricados na programação. Com os resultados obtidos neste estudo é possível afirmar que a Prova de Conceito foi bem sucedida. No entanto, ainda são necessários novos estudos antes de uma afirmação mais forte.

Um outro questionamento que ficou subjacente ao anterior é se o bergsonismo poderia ser utilizado como algo semelhante ao método científico na Ciência da Programação. Embora esta seja a proposta do bergsonismo ao ser aplicado no estudo de objetos metafísicos, ainda é preciso submetê-lo a mais testes antes de sugerir que tenha este mesmo efeito na Ciência da Programação. Embora também seja um pouco antecipado sugerir tais coisas, é possível tecer alguns comentários com base na vivência de realizar este estudo.

Enfim, deve-se ter em mente que este estudo realizou uma prova de conceito de forma que são necessárias maiores investigações antes de sugerir que este é de fato o melhor método de investigação da programação. No entanto, é possível afirmar com os dados apresentados que ele pode ser considerado como um método de investigação possível para o estudo da programação.

5.2.1 Rigor Bergsonista

Um aspecto do bergsonismo que ficou bastante evidente é seu rigor. O rigor bergsonista assemelha-se ao rigor matemático, onde, caso o raciocínio infrinja a um conjunto de

regras, ele não pode ser concluído. A título de exemplo, considere uma divisão onde o programa foi tomado como objeto. Dito isso, o bergsonismo exige que a pessoa se coloque na duração de programa. Dado que duração é uma preocupação no tempo, para retomar a duração do programa é preciso retomar qual sua preocupação em um dado momento. Apenas por esta afirmação já se percebe que a divisão é inconsistente pois um programa, enquanto objeto, não tem nenhuma preocupação. Além disso, para retomar a duração de um programa, a pessoa teria que ter acumulado vivências (memória) atuando como programa. Assim, não há como continuar com a investigação.

Esse mesmo tipo de situação ocorre em diversos níveis da análise. Isso mostra que o resultado obtido pelo bergsonismo não é aleatório mas segue à um conjunto estrito de regras. Em termos de vivência, o bergsonismo equivale ao estabelecimento de uma prova matemática, tanto em suas dificuldades quanto em seu encadeamento. Caso o enunciado bergsonista mostre falhas, tal qual ocorre na matemática ele pode ser refinado até atingir sua forma final. Também, tal qual a uma prova, o fato de se ter conseguido demonstrar o enunciado não significa que ele esteja correto mas apenas que ele é suficientemente confiável. O bergsonismo diferente do que ocorre na matemática não busca partir de um conjunto de premissas e por meio de um processo de inferência chegar no enunciado mas parte de uma intuição e busca, por meio de um processo racional, verificar sua adequação aos princípios bergsonistas.

No entanto, como cada revisão requer um esforço inteiramente novo, o resultado de cada análise acaba sendo sempre um pouco diferente. Inicialmente as variações são incrementais, ou seja, a memória de uma análise propicia que a seguinte seja mais aprofundada. Em um segundo momento a análise começa a resultar em nuanças que se equivalem até que se chegue à um resultado que pareça suficiente. Mais uma vez, este é o mesmo processo que ocorre com as provas matemáticas onde cada prova passa por refinamentos sucessivos até que pareça suficiente. Se diversas pessoas fizerem a mesma prova, cada uma apresentará uma nuança.

5.2.2 Objetividade Bergsonista

O bergsonismo também se propõe a ser objetivo no sentido de que a vivência de um possa ser reconhecida na vivência do outro por meio de um esforço de intuição. Durante o desenvolvimento deste trabalho este aspecto não foi amplamente evidenciado uma vez que não houve uma ampla divulgação de seus resultados. O que houve foi um reencontro contextualizado e qualitativo das idéias trabalhadas na biliografia da área.

Ao subter a descrição da vivência à avaliação de algumas pessoas, percebeu-se que de imediato ou elas se reconhecem na vivência e a aceitam ou elas não se reconhecem e a

rejeitam. Isso é semelhante ao que acontece com a ocorrência dos fenômenos naturais, com base em um experimento, ou o fenômeno é reconhecido e aceito ou não. Em ambos os casos paercebeu-se que não há discussões prolongadas a respeito da vivênica ou fenômeno, ele apenas é reconhecido ou não. Nesse sentido, pode-se dizer que o esforço de intuição que leva uma pessoa a reconhecer ou não uma vivência, equivle ao experimento no método científico. No entanto, isso também requer novas investigações.

Em outras palavras, no método científico, a certeza suficiente vem da reprodutibilidade do experimento. No bergsonismo, o equivalente ao experimento é o esforço de intuição que permite supor a duração de uma vivência. Se diz que este processo equivale ao experimento científico pelo fato da intuição requerer acúmulo de vivências (ou experiência) suficiente para ser considerado um *expert* no assunto. Portanto, para supor uma duração é necessário realizar um experimento mental que pode ser reproduzido por outrem.

5.2.3 Evolução das Teorias Bergsonista

Uma preocupação inerente à um método que se supõe científico é a possibilidade de corrigir, refinar ou refutar hipóteses. Quando isso é possível, se diz ser possível evoluir a ciência. Nesse sentido, o bergsonismo exige atenção para que não se recaia nas dificuldades causadas pela abstração oriundas do hábito do pensamento conceitual. Em outras palavras, ao evoluir uma teoria bergsonista, deve-se utilizar o bergsonismo e não a semiose. Isto é, deve-se ignorar a forma e os conceitos e supor a duração original para depois reecontrar os conceitos e a forma de maneira controlada.

A critica de uma investigação bergsonista que não passe por essas etapas tende-se à perder-se na forma ou nos conceitos e assim contribuir pouco com o desenvolvimento metafísico da área. Assim, a evolução bergsonista pode ser um pouco traiçoeira para quem não tem costume com seus método e isso requer atenção. Da mesma forma, deve-se cuidar para que a crítica não se baseie apenas nos termos reencontrados mas na vivência em si. A crítica dos termos usados no reencontro é importante no sentido de facilitar ou dificultar à intuição supor a duração mas não pode ser o foco principal da crítica.

A crítica bergsonista deve ser sobre a vivência. O nível de percepção pode estar desajustado, por exemplo, a análise bergsonista realizado no capítulo 4 poderia ter sido apresentada para a programação em geral. Isso poderia ser criticado descrevendo e comparando as durações que se interpõe em desenvolvimento de diferentes classes de programas. E assim o estudo poderia ser refinado, localizando a análise para programas acadêmicos. Pode-se também investigar se o tipo de software altera o resultado, por

exemplo, o estudo realizado no capítulo 4 foi conduzido para programas acadêmicos em geral. Ele não considerou se haveria ou não diferença para programas algorítimicos, interativos, *et cetera*.

5.2.4 Sentido dos Resultados Obtidos

Não basta que seja possível derivar resultados por meio de um método. É preciso que esses resultados façam sentido de alguma maneira para que possam ser úteis. Os resultados obtidos da investigação bergsonista sobre a programação puderam ser utilizados para apresentar, ao menos uma, explicação para diversos fenômenos e questionamentos sobre programação que ainda estão em aberto.

Por exemplo, com base na mudança de duração foi possível demarcar a diferença entre análise e programação. Isso mostra que a transição do modelo de análise para o modelo de *design* não é apenas arbitrário mas uma mudança cognitiva que se dá em função do acúmulo de memória sobre o domínio do problema. Além disso, ao consolidar como programaçãodo *design* e codificação, mostrou-se que em programas acadêmicos existe um *continuum* entre os diagramas de alto nível até o código-fonte. Isso sugere que a relação com os diagramas seja orgânica de forma que devam ser utilizados como um apoio cognitivo e não como documentação.

5.2.5 Limite do Bergsonismo

Naturalmente o bergsonismo não é uma panacéia. Ele é um método que pode ser utilizado em estudos metafísicos e ele parece adequado ao estudo metafísico da programação. No entanto, como em qualquer ciência, a Ciência da Programação requer uma metodologia de estudos. Assim, sugere-se que o bergsonismo possa ser utilizado como um dos métodos que compões a metodologia de pesquisa em programação.

O limite do bergsonismo é a duração, isto é, ele não serve para estudar a extensão. Para Bergson, a extensão de fato não importava, no entanto, na programação ela importa e precisa ser abordada de maneira adequada. Sugere-se, por exemplo, que o intuicionismo matemático de Brouwer possa ser utilizado no lado da extensão no bergsonismo. Tal afirmação se deve ao fato de ambos partirem de pressupostos semelhantes e coincidirem em muitos aspectos exceto no fato de que Bergson, enquanto metafísico, se volta à duração e Brouwer, enquanto matemático, se volta à extensão. De qualquer forma, o bergsonismo não pode ser desprezado como ocorre na matemática pois nela a interposição de durações é simples enquanto na programação, complexa.

Em outras palavras, o bergsonismo estuda vivências, ele não estuda objetos e nem fenômenos. Caso deseje-se compor o estudo sobre programação com objetos e fenômenos o bergsonismo precisa ser complementado. Se diz complementado pois a metafísica parece ser um elemento essencial na programação. Considere por exemplo uma investigação sociológica sobre times de programação, mesmo que se estude todas as interações, elas ainda precisarão resultar em um programa e o entendimento sobre programação é resultado de um estudo metafísico.

5.3 Contribuições deste Estudo

Embora a obra de Bergson seja "simples", ela é complexa no sentido de que requerer que o leitor adote um paradigma que inverte o hábito de seu pensamento. Até que se perceba isso, a leitura do bergsonismo se torna obscura. No entanto este ponto não é apresentado de maneira expressiva nem por Bergson e nem por Deleuze. Assim, uma contribuição deste estudo é apresentar este ponto de maneira clara e direta, facilitando assim o entendimento sobre o bergsonismo.

Este estudo também apresentou e criticou uma série de abordagens metodológicas que vem sendo utilizadas no estudo da programação. Como foi sugerido, até agora elas não mostraram resultados suficientes, cada uma por um motivo diferente. Assim, uma outra contribuição deste estudo foi relacionar tais abordagens metodológicas com as dificuldades apontadas pelo SEMAT. Além disso sugeriu-se também que o SEMAT está tentando resolver os problemas que apontou usando as mesmas abordagens metodológicas que os criaram.

Uma outra contribuição deste estudo foi mostrar que a idéia de *ciência*, em programação, foi uma arbitrariedade histórica e que o uso de ciência no sentido estrito da palavra não se aplica. Isso se justifica, pois uma série de pesquisas foram direcionadas pela idéia de que a programação fosse uma ciência e acabaram chegando em resultados pouco expressivos. Naturalmente, o termo Ciência da Programação pode ser utilizado, desde que se tenha claro que o termo *ciência* na expressão tem sentido *lato*.

Outra contribuição deste estudo foi ressaltar a importância da metafísica na programação. Com base nos resultados obtidos por este estudo, a metafísica parece ser o centro da Ciência da Programação tal qual o fenômeno é o centro da Ciência Natural. Assim, em Ciência da Programação, seja a investigação científica ou epistemológica ela deverá referir-se à metafísica para ter algum significado concreto ou se justificar. Considere, por exemplo, uma investigação epistemológica onde dois métodos de programação são comparados. Estudar seu resultado ou os métodos em si não leva a um resultado conclusivo, é preciso investigar o que ocorre, em termos de consciência, para

verificar se os métodos suscitam ou não mesma duração. O mesmo vale para investigações científicas, pois a investigação de um código-fonte deve referir-se ao que ocorre em termos de duração, do contrario, o programa se mostra como um conjunto de símbolos.

Além dessas, uma outra contribuição foi apresentar um método que capaz de investigar objetos metafísicos de forma similar ao científico. O método científico é o elemento principal da ciência, caso espere-se desenvolver uma Ciência da Programação será necessário o uso de um método que possa ser considerado científico. O bergsonismo se mostra como um método que tem tais características e que pode ser aplicado sobre objetos metafísicos. Isso possibilita que a programação seja tratada como uma "Ciência Metafísica".

Este estudo apresentou também uma caracterização da programação mostrando-a como algo distinto da análise e teste. Isso propicia um critério de demarcação entre essas atividades, possibilitando, por exemplo, esforços de formação direcionados para cada uma dessas atividades. Além disso, a consolidação do *design* e implementação em uma única atividade propicia uma maior compreensão sobre como ocorre a programação. Isso, por sua vez, propicia o refinamento das ferramentas, técnicas e princípios de programação que estejam consoantes com a nuança do momento.

Ainda foi possível explorar a essência e dinâmica da programação. Com isso se torna possível compreender, por um lado, quais intuição se deseja despertar durante a programação. Isso pode ser direcionado por meio de situações de aprendizagem que criem memória de interesse. Por outro lado, é possível compreende qual tipo de razão é útil no desenvolvimento de software ajudando também a direcionar a formação profissional.

5.4 SEMAT

A crise do software pode ser considerada a origem de diferentes problemas. O [Jacobson *et al.*, 2009] aponta alguns deles, como a grande variedade de técnicas muito parecidas com diferenças pouco entendidas; a falta de uma base confiável para validação de experimentos; e a separação entre a pesquisa e a prática da programação.

O problema da existência de diferentes métodos muito parecidos, com diferenças pouco entendidas, pode ser tratado pelo bergsonismo, ao considerar que a base de investigação é a consciência e sua relação com a verdade enquanto experiência. No caso específico do problema apresentado, para identificar se duas técnicas são diferentes, é preciso avaliar, em termos da experiência, se ela produzem durações, nuanças ou direcionamentos mentais distintos. Caso apresentem, pode-se dizer que sejam diferentes. Do contrário, a diferença é apenas na forma, sendo iguais em essência.

O problema da falta de uma base confiável para validação de experimentos também pode ser tratado pelo bergsonismo. Este propõe que o tentame seja a experiência com o real e a validação seja a manifestação desta experiência em forma de intuição. Neste sentido, o bergsonismo apresenta uma alternativa de validação que, embora filosófica, pode ser melhor adequada à programação do que outras.

O problema da lacuna entre a pesquisa e a prática da programação tende a desaparecer com o bergsonismo. Como a validação é com base na experiência, a investigação e proposta de teorias também requer experiência como base. Este ponto é o que traz rigor ao bergsonismo, ou melhor, é a forma pela qual o bergsonismo impõe rigor. Se não houver um esforço de intuição, que leve a um contato com o real e à suposição de uma duração, a tentativa de compreensão do objeto incorre em constantes contradições, que podem resultar em uma explicação artificial ou ingênua. Assim, as proposições bergsonistas se assemelham à uma prova matemática. Naturalmente, assim como ocorre na matemática, o rigor não é absoluto; podem haver explicações robustas e frágeis, além do equivalente dos lemas e corolários.

5.5 Objeções ao Intuicionismo

Tanto o bergsonismo como o brouwerismo sofreram grande oposição em sua área de origem. Embora seja possível enumerar e responder à uma série de críticas feitas ao bergsonimo [Russel, 1912] [Ponty, 2006] e ao brouwerismo [Reid, 1996], sugere-se que a oposição, como muitas vezes ocorre nas ciências *cf.* [Kuhn, 2005], nasceu por um viés político e não técnico [Prado, 1999] e [da Costa, 1992]. Isso fica claro ao considerar como exemplo o comentário de Hilbert afirmando que o brouweismo removeria da matemática suas mais belas conquistas. No bergsonismo foram dois os problemas, o primeiro foi sua necessidade de subverter o hábito do pensamento conceitual para apresentar suas teses; o segundo foi sua tentativa de subverter a forma com que se conduz as pesquisas em metafísica buscando, por meio do aumento da precisão, a unificação de todos os sistemas. No entanto, as críticas ao bergsonismo se deram por meio do pensamento conceitual, tendo por base a forma com que se normalmente conduz as pesquisas em metafísica.

No fim, esses problemas podem se resumir ao fato de que esses paradigmas não respondiam, ao menos da forma que se gostaria, às inquietações dos pesquisadores na época em que foram propostas [da Costa, 1992]. Neste sentido, este pode ser também o principal ponto fraco da teoria proposta neste estudo.

Capítulo 6

Conclusão

Este trabalho discutiu a possibilidade de usar o bergsonismo como método de estudo na Ciência da Programação. Para explorar esta possibilidade, apresentou uma prova de conceito. Embora ainda sejam necessários novos estudos para determinar com precisão se o bergsonismo efetivamente pode ou não ser utilizado, ou melhor, se ele é um método adequado e útil no estudo na Ciência da Programação, pode-se dizer que ele foi bem sucedido enquanto prova de conceito.

Como não poderia deixar de ser, em uma prova de conceito, muitos elementos importantes foram desprezados, os quais em estudos subsequentes deverão ser cuidadosamente retomados e explorados. Um ponto a ser refinado é o estudo da programação em si. Na prova de conceito, se assumiu como objeto as durações tradicionais de análise, *design*, implementação e testes. Em um estudo real, se deve questionar essas durações e verificar se as durações da vivência correspondem com essas durações.

Outro ponto de melhoria, consiste em verificar se existe diferença cognitiva na programação de classes e tipos de programas diferentes. A prova de conceito assumiu apenas o software acadêmico. Este estudo não entrou no mérito das diferenças entre programas acadêmicos e comerciais, de pequeno e grande porte ou algoritmo e interativo. Essas são questões que devem ser exploradas para uma melhor compreensão da programação.

Em termos de método de pesquisa, ainda é cedo para afirmar se ele se mostrou melhor ou pior do que os outros métodos, atualmente utilizados no estudo da programação. Pode-se dizer no entanto que ele apresentou resultados diferentes. Convém também, em estudos subsequentes, explorar com mais detalhes os resultados obtidos e compará-los com os de outros métodos, para entender qual a melhor aplicação para cada um deles.

Uma ciência não é estudada por um único método, mas por uma metodologia.

Assim, é possível afirmar que o bergsonismo pode compor a metodologia de estudo da programação. Se diz compor, pois ele trabalha os aspectos metafísicos mas não os matemáticos, sociais, *etc.* Percebeu-se, daí, que a programação é uma atividade humana complexa, que evolve diferentes dimensões e que precisam ser melhor compreendidas para fazer avançar essa ciência.

Talvez, a maior contribuição apresentada por este estudo tenha sido sugerir que a metafísica tem papel fundamental na programação. Diz-se a maior contribuição pois, em geral, a metafísica é ignorada e caso ela de fato se mostre um elemento central, ignorá-la atrasaria o desenvolvimento da Ciência da Programação. Embora também seja cedo para tal afirmação, o fato é que quando se considera a metafísica, problemas reconhecidamente intricados começam a ser resolvidos.

Referências Bibliográficas

[Beynon *et al.*, 2008] Beynon, M. Russell, B. Chan, Z. E. (2008). Intuition in Software Development Revisited. *20th Annual Psychology of Programming Interest Group Conference*, Lancaster, UK.

[Bergson, 1910] Bergson, H. (1910). *Time and Free Will*. George Allen & Unwin Ltd, London.

[Bergson, 1911] Bergson, H. (1911). *Matter and Memory*. George Allen & Unwin Ltd., London.

[Bergson, 1920] Bergson, H. (1920). *Mind-Energy: Lectures and Essays*. Henry Holt and Company, NY.

[Bergson, 1922] Bergson, H. (1922). *Creative Evolution*. MacMillan and co., London.

[Bergson, 1980] Bergson, H. (1980). *O Riso ensaio sobre a significação do comico*. Zahar.

[Bergson, 2006] Bergson, H. (2006). *Duração e Simultaneidade*. Martins Fontes.

[Bergson, 2006] Bergson, H. (2006). *O Pensamento e o Movente: Ensaios e Conferências*. Trad. Bento Prado. Martins Fontes.

[Bertalanffy, 2008] Bertalanffy, L. v. (2008). *Teoria geral dos sistemas: fundamentos, desenvolvimento e aplicações*. Vozes.

[Boehm & Sullivan, 2000] Boehm, B. Sullivan, S. (2000). Software Economics: A Roadmap. *ICSE '00 Proceedings of the Conference on The Future of Software Engineering*. ACM.

[Booch *et al.*, 2007] Booch, G. Maksimchuk, R. A. Engle, M. W. Conallen, J. Houston, K. A. Young, B. J. (2007). *Object-Oriented Analysis and Design with Applications*. Pearson Education, terceira edição.

[Booch, 2010] Booch, G. (2010). Architecture as a Shared Hallucionation. *IEEE Software*.

[Brooks, 1995] Boorks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley Professional.

[Brouwer, 2011] Brouwer, L. E. J. (2011). *Brouwer's Cambridge Lectures on Intuitionism*. Cambridge University Press, reimpressão do original publicado em 1981.

[Cerqui, 2002] Cerqui, D. (2002). The future of humankind in the era of human and-computer hybridization: An anthropological analysis. *Ethics and Information Technology*, v.4 i.2. Kluwer Academic Publishers Hingham.

[Cohen, 2000] Cohen, R. A. (2000). Ethics and cybernetics: Levinasian reflections. *Ethics and Information Technology*, v.2 i.1. Kluwer Academic Publishers Hingham.

[Conkling & Christensen, 2009] Conkling, J. Christensen, K. (2009). Building Shared Understanding of Wicked Problems. *Rotman Magazine*, winter. University of Toronto.

[da Costa, 1992] da Costa, N. C. A. d. (1992). *Introdução aos Fundamentos da Matemática.* Hucitec.

[Deleuze, 1986] Deleuze, G. (1986). *Cinema 1: The Movement-Image*. Univ Of Minnesota Press.

[Deleuze, 1996] Deleuze, G. (1996). *O Atual e o Virtual*. Flammarion, Paris.

[Deleuze, 1999] Deleuze, G. (1999). *Bergsonismo*. Coleção TRANS. Editora 34, traduzido por Orlandi, L. B. L.

[Deleuze & Guattari, 2010] Deleuze, G. Guattari, F. (2010). *O que é a Filosofia?*. Editora 34.

[DeMarco, 1986] DeMarco, T. (1986) *Controlling Software Projects: Management, Measurement, and Estimates*. Prentice Hall.

[Dupont, 2013] Dupont, C. (2013). *Phenomenology in French Philosophy: Early Encounters*. Springer Science & Business Media.

[Dijkstra, 1976] Dijkstra, E. W. (1976). *A Discipline of Programming*. Prentice-Hall.

[Dijkstra, *s.d.*] Djkstra, E. W. (*s.d.*). *Programming Considered as a Human Activity*. EWD117. *In.* E. W. Dijkstra Archieve.

[Dijkstra, 1969] Dijkstra, E. W. (1969). *Notes on Structured Programming.* T.H.-Report 70-WSK-03. Technische Hogeschool Einhoven.

[Dreyfus, 1998] Dreyfus, H. L. (1998). *A Phenomenology of Skill Acquisition as the basis for a Merleau-Pontian Non-representationalist Cognitive Science* University of California, Berkeley.

[Feyerabend, 2010] Feyerabend, P. (2010) *Adeus à Razão.* Unesp.

[Flammarion, 1913] Flammarion, E. (Ed.) (1913). *Le matérialisme actuel.* E. Flammarion, Paris.

[Foote & Yoder, 1997] Foote, B. Yoder, J. (1997). Big Ball of Mud. *Fourth Conference on Patterns Languages of Programs (PLoP '97/EuroPLoP '97)*

[Guy & Champagnat, 2012] Guy, O. Champagnat, R. (2012). Flashback in interactive storytelling. *ACE'12 Proceedings of the 9th international conference on Advances in Computer Entertainment.*

[Gödel, 1990] Gödel, K. (1990). *Kurt Gödel: Collected Works, volume II — Publications 1938-1974.* Ed: Feferman, S. Oxford University Press.

[Hansen, 2004] Hansen, M. B. N. (2004). *New Philosophy for New Media.* The MIT Press.

[Haynes, 2001] Haynes, J. D. (2001). Churchman's Hegelian Inquiring System and Perspectival Thinking. *Information Systems Frontiers*, v.3 i.1. Kluwer Academic Publishers Hingham.

[Harel, 1987] Harel, D. (1987). Statecharts: a visual formalism for complex systems *Science of Computer Programming 8.*

[Heidegger, 1977] Heidegger, M. (1977). *The Question Concerning Technology and Other Essays.* Garland Publishing, Inc.

[Heyting, 1971] Heyting, A. (1971). *Intuitionism: An Introduction.* 3a ed., Noth-Hollnd Pub. Co., Amsterdam.

[Hintikka, 2003] Hintikka, J. (2003). The notion of intuition in Husserl. *Revue internationale de philosophie*, n. 224.

[Humphrey, 2005] Humphrey, W. S. (2005). *PSP: A Self-Improvement Process for Software Engineers.* Addison-Wesley Professional.

[Jackson, 2005] Jackson, M. (2005). Problem Frames and Software Engineering. *Information and Software Technology*, special issue on the 1st International Workshop on Advances and Applications of Problem Frames. v. 47 n. 14.

[Jackson, 2013] Jackson, M. (2013). Formalism and Intuition in Software Engineering *In. Perspectives on the Future of Software Engineering: a Festschrift in Honour of Dieter Rombach*. Springer verlag.

[Jacobson, 1995] Jacobson, I. (1995). *The Object Advantage: Business Process Reengineering with Object Technology*. Addison-Wesley.

[Jacobson & Meyer, 2009] Jacobson, I. Meyer, B. (2009). Methods Need Theory. *Dr. Dobb's*.

[Jacobson & Spence, 2009] Jacobson, I. Spence, I. (2009). Why we Need a Theory for Software Engineering. *Dr. Dobb's*.

[Jacobson *et al.*, 2009] Jacobson, I. Meyer, B. Soley, R. (2009). The SEMAT Initiative: A Call for Action. *In.www.drdobbs.com*, visto em 2013.

[Jacobson *et al.*, 2013] Jacobson, I. Ng, P. W. McMahon, P. E. Spence, I. Lidman, S. (2013). *The Essence of Software Engineering: Applying SEMAT Kernel*. Addison-Wesley.

[Johns & Saks, 2004] Johns, G. Saks, A. M. (2040). *Organizational Behavior: Understanding and Managing Life at Work* Pearson Prentice Hall, sexta edição.

[Johnson *et al.*, 2012] Johnson, P. Ekstedt, M. Jacobson, I. (2012). Where's the Theory of Software Engineering? *IEEE Software*, v. 29 n. 5.

[Kant, 1995] Kant, I. (1995). *Crítica da faculdade do juízo*. Biblioteca de filosofia. Forense Universitária.

[Knuth, 1974] Knuth, D. E. (1974). Computer Programming as an Art. *Communications of the ACM*, v. 17 n. 12.

[Kuhn, 2005] Kuhn, T. (2005). *A Estrutura das Revoluções Científicas*. Volume 115 de Coleção Debates. Perspectiva.

[Lawlor & Moulard, 2013] Lawlor, L. Moulard, L. (2013). Henri Bergson. *The Stanford Encyclopedia of Philosophy*.

[Lehar, 2003] Lehar, S. M. (2003). *The World in Your Head: A Gestalt View of the Mechanism of Conscious Experience*. Psychology Press.

[Marcondes *et al.*, 2009] Marcondes, F. S. Montini, D. A. Vega, I. S. Dias, L. A. V. (2009). Elicitação da dificuldade de entendimento com base na teoria de Bergson. *Anais do IV WORKSHOP DE PÓS-GRADUAÇÃO E PESQUISA DO CENTRO PAULA SOUZA*.

[Marcondes *et al.*, 2011] Marcondes, F. S. Vega, I. S. Dias, L. A. V. (2011). An approach for modeling a formal Use Case Type at early development phase without loosing abstraction. *Innovations in Systems and Software Engineering (Print), v. 7*. Springer.

[Martin, 2003] Martin, R. C. (2003). *UML for Java Programmers*. Prentice Hall.

[Martins, 1991] Martins, F. R. (1991). *Estrutura de uma floresta mesófila*. Editora da Universidade Estadual de Campinas.

[McConnell, 2004] McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press.

[Miller *et al.*, 2010] Miller, F. P. Vandome, A. F. John, M. (2010). *Meta-Object Facility*. VDM Publishing.

[Miyoshi, 2004] Miyoshi, H. (2004). From Reflection to Interaction: An Indirect Approach to the Philosophy of Computation. *CRPIT '03 Selected papers from conference on Computers and philosophy*, v.37.

[Morris & Brown, 2014] Morris, W. E. Brown, C. R. (2014). David Hume. *The Stanford Encyclopedia of Philosophy*.

[Naur & Randell, 1969] Naur, P. Randell, B. Eds. (1969). *Software Engineering*. NATO Science Committee.

[Naur, 1985] Naur, P. (1985). Intuition in Software Development. *Lecture Notes in Computer Science*, v. 186. Springer.

[Øhrstrøm, 2010] Øhrstrøm, P. (2010). Towards a common language for the discussion of time based on prior's tense logic. *COST'10 Proceedings of the 2010 international conference on Multidisciplinary Aspects of Time and Time Perception*, Springer.

[Papineau, 1996] Papineau, D. (1996). *The Philosophy of Science*. Oxford Readings in Philosophy.

[Peirce, 2012] Peirce, C. S. (2012). *Philosophical Writings of Peirce*. Buchler, J. Courier Corporation.

[Poincaré, 1905] Poincaré, H. (1905). *Science and Hypothesis*. The Walter Scott Publishing Co, NY.

[Poincaré, 1907] Poincaré, H. (1907). *The Value of Science*. The Science Press, NY.

[Poncaré, 2012] Poincaré, H. (2012). *Science and Hypothesis*. Courier Corporation.

[Polya, 1957] Polya, G. (1957). *How to Solve It: A new Aspect of Mathematical Method.* Princeton University Press., segunda edição.

[Pombo, 2012] Pombo, O. (2012). Conceptions of intuition in Poincaré's philosophy of mathematics. *Faculty of Sciences, University of Lisbon.*

[Ponty, 2006] Ponty, M. (2006). *Fenomenologia da Percepção*. Martins Fontes.

[Popper, 2004] Popper, K. R. (2004). *A Lógica da Pesquisa Científica*. Editora Cultrix.

[Posner, 1993] Posner, M. I. (1993). *Foundations of Cognitive Science*. Bradford.

[Prado, 1999] Prado, B. (1999). A filosofia seminal de Bergson. *Jornal Folha de São Paulo.*

[Rapaport, 2005] Rapaport, W. J. (2005). *Philosophy of Computer Science: An Introductory Course*. State University of New York at Buffalo, Buffalo, NY.

[Reale, 2002] Reale, G. (2002). *O saber dos antigos*. Loyola.

[Reid, 1996] Reid, C. (1996). *Hilbert*. Copernicus Series. Springer Science & Business Media.

[Rittel & Webber, 1973] Rittel, H. W. J. Webber, M. M. (1973). Dilemmas in a General Theory of Planning. *Policy Sciences*. n. 4. Elsevier.

[Resconi & Nikravesh, 2008] Resconi, G. Nikravesh, M. (2008). Morphic computing. *Applied Soft Computing*, v.8 i.3.

[Robbins, 2002] Robbins, S. E. (2002). Semantics, experience and time. *Cognitive Systems Research*, v.3 i.3. Elsevier Science Publishers B. V. Amsterdam.

[Runenson & Höst, 2009] Runenson, P. Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, v. 14, i. 2, Springer.

[Russel, 1912] Russel, B. (1912). The Philosophy of Bergson. *The Monist*. v. 22 n. 3.

[Shaw, 1990] Shaw, M. (1990). Prospects for an Engineering Discipline of Software *IEEE Software.*

[Silva, 2010] Silva, G. C. R. F. (2010). O Método Científico na Psicologia: Abordagem Qualitativa e Quantitativa. *In.psicologia.com.pt*, visto em 2014.

[Sipser, 2012] Sipser, M. (2012). *Introduction to the Theory of Computation* Cengage Learning.

[Stacy & MacMillan, 1995] Stacy W. MacMillan, J. (1995). Cognitive Bias in Software Engineering *In. Communications of the ACM.* v. 38 n. 6.

[Stanovich & West, 2000] Stanovich, K. E. West, R. F. (2000). Individual differences in reasoning: Implications for the rationality debate? *BEHAVIORAL AND BRAIN SCIENCES*, n. 23.

[Sørensen & Urzyczyn, 2006] Sørensen, M. H. Urzyczyn, P. (2006). *Lectures on the Curry-Howard Isomorphism.* Volume 149 de Studies in Logic and the Foundations of Mathematics. Elsevier.

[Suppe, 1998] Suppe, F. (1998). Understanding Scientific Theories: An Assessment of Developments, 1969-1998. *Philosophy of Science*, v. 67. The University of Chicago Press.

[Teixeira, 2001] Teixeira, L. (2001). *A Doutrina dos Modo de Percepção e o Conceito de Abstração na Filosofia de Espinosa.* UNESP.

[Turing, 1937] Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society.* s. 2 v. 42.

[Turing, 1950] Turing, A. M. (1950). Computing Machine and Intelligence. *Mind.* n. 59.

[Turner, 2014] Turner, R. (2014). The Philosophy of Computer Science. *The Stanford Encyclopedia of Philosophy.*

[Vega, 2012-2015] Vega, I. S. (2012-2015). Notas de Aula. *Grupo de Estudos em Modelagem de Software (GEMS).* Pontifícia Universidade Católica de São Paulo.

[Voegelin, 2006] Voegelin, A. (2006). Sonic memory material as 'pathetic trigger'. *Organised Sound*, v.11 i.1.

[Weinberg, 1998] Weinberg, G. M. (1998). *The Psychology of Computer Programming.* Dorser House Pub., reimpressão do original publicado em 1971.

[Wegner, 1997] Wegner, P. (1997). Why Interaction is more Powerful than Algorithm. *Communications of the ACM.* v. 40 n. 5.

[Yin, 2015] Yin, R. K. (2015). *Estudo de Caso*. Bookman.