表紙

- 提出年度
  2012年度
- テーマ
  ソフトウェアプログラム可食化に関する研究
- 副題
  スパゲッティへの具現化とその枠組みの提案
- 学部学科名
  コンピュータ理工学部 ネットワークメディア学科
- 学生証番号
  947424
- 氏名
  宮崎 雅文（Miyazaki Masafumi）
- 指導教員名
  青木 淳 教授

要旨

本研究は、情報表現手段としての「可食化」の提案、ならびに、ソフトウェアメトリクスのための新しい枠組みの提案を目的とするものであり、また本稿は、その研究成果についてまとめあげた論文である。

ソフトウェアプログラムの様々な特徴を計測・評価する行為であるソフトウェアメトリクス。 その行為を自動化する道具（メトリクスツール）は既に多く出まわっており、計測対象（プログラム）の各特徴を数値やグラフとして表し、客観的かつ定量的な情報を提供してくれる。 しかしながら、ソフトウェアプログラムは複雑な構造物である。実際に計測しようにも、その指標には把握しきれないほどの種類が存在するため、具体的にどの指標を用いて何を目安に計測すれば良いのか分かりづらい面がある。

一方、メトリクス結果を三次元グラフィクスの都市として表現する先行研究[1]が存在する。 ソフトウェアプログラムの規模や構造を上手く「可視化」するその手法は、数値で示されるよりも遥かに直感的な理解を与えてくれた。 私はこの研究に大きな感銘を受けたのだが、同時にとある疑問を抱き、その疑問に端を発し着想したのが「可食化」である。

長年の田舎育ちのためか、都市（ビルの群）というものは私にとって身近な存在ではなく、ソフトウェアプログラムのメタファ（隠喩表現）としての都市にも、あまり身近さを感じなかった。 故に「どうして都市として表現したのだろうか。もっと分かりやすい別のメタファはないものか...」という疑問を抱くことになる。 その解を求めるべく幾つかの連想を経た後、かの「スパゲッティプログラム」という言葉に辿り着き、メタファとしての「スパゲッティ」ひいては「情報の可食化」を考え始めたのであった。 数値による可視化に満足せずグラフィクスによる可視化が登場したのであれば、食べ物による可食化が登場することにも何ら不思議はないはずである。

そこで、本研究では「スパゲッティを題材としたソフトウェアプログラムの可食化」を具体例として示し、その有効性を実証することによって、新しい情報表現手段としての「可食化」を提案する。 また、その実現のために新しくソフトウェアメトリクスツールを用意したが、これは既存のツールとは異なり、より利用者の要求に即した情報提供を行うための特徴を備えている。 本稿ではその特徴を、ソフトウェアメトリクスのための新しい枠組みとして抽象化し、別途提案することにする。

学生証番号：947424　氏名：宮崎 雅文

目次

学生証番号：947424　氏名：宮崎 雅文

はじめに

身近なものに対する疑問が研究の発端になるとはよく言ったものだが、自身の取り組む研究として相応しいテーマはそう簡単に見つかるものではない。 学部4回生としての研究に際して、そのテーマの採択ほど時間を要することは他になかったであろう。 長きに渡って悩み考えあぐねた末にようやく「スパゲッティ」という着想を得て、もうこれ以外に道（発想できること）はない...と思ったほどである。

> ソフトウェアプログラムをスパゲッティにしてしまおう

ある先行研究[1]を師にご教示いただいたことでこの主題に至ったのだが、その経緯の詳説は後の章に譲るとして。 まずはソフトウェアプログラムのメタファ（隠喩表現）となる「スパゲッティ」の調査に奔走した。 日夜、数々のスパゲッティを食べ歩き、また調理し、その歴史や性質を徹底的に調べ上げた結果、私は「スパゲッティ」という食材が織り成す広大な世界を垣間見たのだ。 そんな私には、どうにも納得できないことが一つあった。

研究テーマの着想には、かつてのソフトウェア開発における問題でもあった「スパゲッティプログラム」という言葉が一役買ったのだが、様々な文献を当たってみると、その全てにおいて悪い意味（手のつけようのない複雑なプログラムという意味）で用いられていたのだ。 ピンからキリまである多様なスパゲッティを食してきた身として、「スパゲッティ」という言葉を一概に悪い意味で使っていることに対して腹立ちをも覚えたほど。 このことについて、本研究の中間発表で次のように述べたことがある。

> *世界中のイタリアンシェフを敵に回すような表現であり、言語道断である。プログラムと同様にスパゲッティにも良し悪しはあるのだ。*

自分のことながらその荒ぶり方には少し呆れるが、この「スパゲッティへの情熱」こそが、本研究に取り組むための原動力となっていたことは相違ない。 かくして、スパゲッティとして可食化するための研究が進んでゆくのだが、その結末や如何に。上手く説明できていないが、関心や興味をお持ちいただけたとすれば幸いである。

三つの興味

研究への取り組みとは無関係に、私はある三つのことに興味を抱いていた。ただ、同じ研究活動なら興味のあることに取り組みたいもの。 なるべく興味の対象全てを取り扱うように配慮しながら本研究に取り組んできたので、それぞれを簡単に紹介させていただきたい。

メトリクスに関すること

何かしらの尺度・指標で対象を計測することを「メトリクス」と言うのだが、これは、対象を捉えるための別の側面（切り口）を見出す際に有効な手段である。 同じ対象を見ていたとしても、捉え方（尺度・指標）が異なれば、その見え方も変わるもの。 都合の良い捉え方を見出したり、驚くような新しい発見を促したり、その奥深い世界に広がる大きな可能性を思うと、興味を持たずにはいられない。 そのため、取り組む研究の主題として、ソフトウェアプログラムのメトリクスにまつわるものを採用した。

教育に関すること

元々、自分の知っている情報を誰かに提供するということが好きだった私は、以前から教育系に興味を持っていた。 さらに、上級生として下級生の質問に対応する制度「寺子屋」において、指導側としての役どころを授かっていたこともあり、研究も教育にまつわる内容にできればと考えていた。 そのため、可食化の例題として「初学者のプログラムの良し悪し」に関するテーマを設定し、プログラミング初学者の教育に貢献できるよう構成した。

言語処理系に関すること

直属の先輩である水野 信さんは、ご自身の研究でプログラミング言語の開発をなさったのだが、そのことによって受けた影響が大きい。 予てからプログラミング言語自体に興味を抱いていた私は、水野さんの実例を目にすることで、言語設計や処理系実装の魅力に染められたのであった。 そのため、本研究で用意するメトリクスツールを実現するため、新しい言語「SynapseScript」を設計及び実装することにした。

本文

学生証番号：947424　氏名：宮崎 雅文

第1章 序論

本章では、論旨ごとにその背景や最終的なねらいを明らかにする。

「可食化」の提案

背景

何かしらの指標に基づいてソフトウェアプログラムの性質を計測する「メトリクスツール」というものが存在する。コードの行数や関数の宣言数など、ツールによっては実に数十〜数百種の指標を取り扱うことができる。 そうして計測した種々の指標値は、図1-1のようにグラフとして提示されるか、図1-2のように一覧形式で提示されることが多い。

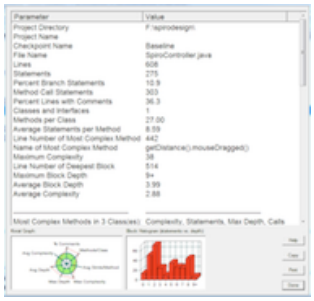


図1-1 一覧表示とグラフ表示の例（SourceMonitor[6]）

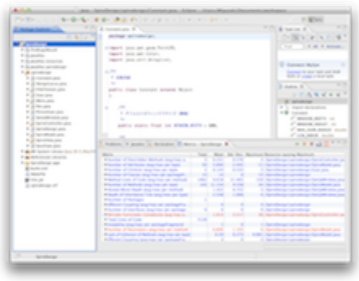


図1-2 一覧表示の例（EclipseMetricsPlugin[7]）

各指標の意味をよく理解し活用できるユーザであれば、数値やグラフといった情報のみでプログラムの良し悪しを判断できるかもしれないが、そうでないユーザならどうだろう。 結果として得られた指標値をどう捉えれば良いのか、さらに、どの指標値に注目すると良いのかすら分別が付かないものである。つまり、従来のメトリクスツールは、計測の対象となる分野に疎いユーザに対して放任的なのである。

ここで、一つの関連研究を紹介しよう。ソフトウェアプログラムの規模や構造を三次元グラフィクスの都市（ビル群）として可視化するものである。（cf. 文献[1]） 「数値」ではなく「都市」で表現することにより、その規模や構造について直感的な理解を促すことができるのだ。

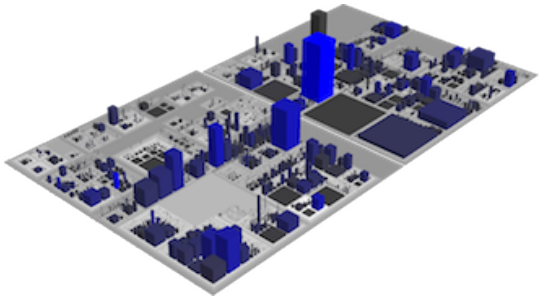


図1-3 都市として可視化する例（CodeCity[11]）

プログラムの性質を表現するメタファ（隠喩表現）が「数値」から「都市」に変化したことで、その捉えやすさは大きく変化した。ならば、これと同様にまた別のメタファを用いることで、さらに直感的な提示方法を提案できないものだろうか。

着想とねらい

次章で詳説する経緯により、上記の背景に対して次のような着想を得た。

> ソフトウェアプログラムを「スパゲッティ」というメタファで表現して、情報の「可食化」を実現する。

背景で述べた通り、従来のメトリクスツールは直感的な理解に向かず、計測対象の分野についての知識がなければ良し悪しを判断することができない。 また、都市として可視化することで直感的な理解を促せるようになったが、それもまだ仮想空間の域を脱しておらず、改善の余地が認められる。 そこで、仮想を超えた現実空間における「スパゲッティ」としてソフトウェアプログラムを可食化し、既存のツールよりもさらに直感的な情報提示を実現したい。

また、上記の着想に対するねらいは次のとおりである。

> 可食化が有効な情報表現手段であることを実証し、その新たな活用の可能性を見出すこと。

本研究では、この「スパゲッティへの可食化」に対する評価実験を執り行い、従来の可視化と比較して直感的な理解が得られるか否かを検証する。 かつ、その有効性を実証し、情報表現手段としての「可食化」を提案することが本研究のねらいである。

本稿での取り扱い

これは本研究における最大の論旨であるため、その論は、本稿全体を通して主張することにする。（第2章 理論と実施計画 - 理論が、その説明の中心となる。）

メトリクスの新しい枠組みの提案

背景

先の可食化を実現するためのツール、つまりソフトウェアプログラムを入力としてスパゲッティを出力する「新しいメトリクスツール」を用意しなければならない。 より具体的には、プログラムからその指標値を算出し、それをスパゲッティのある指標値として写像する（対応付ける）ツールを開発する。 このツールの機能は、「プログラムから各種指標値を得ること」と「それら指標値からスパゲッティの指標値を得ること」の、二段階の写像によって構成される。 また、それぞれの段階もさらに小さな写像によって構成されるため、「写像の仕組み」さえ用意できれば可食化を実現できるのだ。 （捉えようでは「調理」も写像と言えるだろう。どちらにせよ大切なことは、全て写像で表現できるという構造である。）

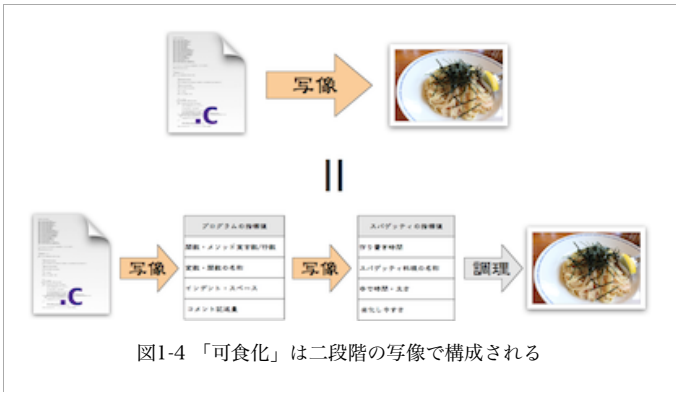


図1-4 「可食化」は二段階の写像で構成される

さて、ここで『ソフトウェア作法』という文献から得た学びを引用しよう。（cf. 文献[2]） 何かしらのソフトウェアを拵える際、一枚岩のようなひとかたまりのプログラムで構成するのではなく、小さな部品を組み合わせることによって構成しよう、というもの。 小さな部品で構成しておけば、その単体の部品を別の場面で再利用できるため、将来の徒労（重複した機能の実装）を減らせるのだ。 その意味で、既存のメトリクスツールは「一枚岩のようなプログラム」であり、再利用性が極めて低く、決して良い例とは言えない。 可能であれば、本研究で開発する「可食化のためのメトリクスツール」は、再利用性の高いソフトウェアとして仕上げたい。

着想とねらい

背景に対する着想は以下の通りである。

> 写像を担う「メトリクスプラグイン」とその繋ぎ役とで構成される「メトリクスグラフ」によって、再利用性の高いメトリクスツールを実現する。

従来は再利用性に欠けていたメトリクスツールも、小さな部品で構成することによりそれを高めることができる。 その効用として、部品同士の組み合わせ次第では、より大きな目的（複雑なメトリクス）も成し遂げられるようになろう。 従来通り数値・グラフ・都市で表現するメトリクス

も、今回のようにスパゲッティで表現するメトリクスも、さらにユーザ自身が考え出す新しいメトリクスでさえも、この「メトリクスグラフ」の構造によって自在に実現できるのである。

この着想に対するねらいを提示する。

> メトリクスツールのための新しい枠組みを提案し、メトリクスツールの改善によってユーザの活用を促進させること。

これまでは、多様なユーザの要求に対応すべくあらゆる指標値を一覧表示し、過剰な情報提示となっていた。 しかし、この枠組み（「メトリクスグラフ」の構造）を採用することで、ユーザが本来求めている指標値に限定して効率的に情報を提示できるのだ。 （特に、複数の指標値を加味して評価を行う場合には、この枠組みの利用による効果が大きい。）

余分な情報を排斥して（冗長性を改善して）メトリクスツールの価値を享受しやすくすること。 また、小さな部品で構成することにより、自由自在なメトリクス（高い適応性）を実現すること。 この二点の相乗効果によってツールの利便性を向上させ、より多くのユーザにツールを活用してもらうことが本研究のねらいである。

本稿での取り扱い

これは本研究における副次的な論旨であるため、第2章 理論と実施計画 - 理論（写像）を中心に、その論を展開することにする。

学生証番号：947424　氏名：宮崎 雅文

第2章 理論と実施計画

本章では、論旨を支える理論とその論旨を主張するための計画について述べる。

理論

可食化で何を表現するか

可食化とは、何かしらの対象物を食べ物として表現することである。 本研究において、この対象物とは「ソフトウェアプログラム」を意味するが、そのプログラムのどのような性質を食べ物に変換するというのか。 つまり、可食化という手法で何を表現しようとしているのか、そのテーマを決めておかねばならない。

今回、可食化の例題として取り扱うテーマは、以下の通りである。

> プログラミング初学者が書くプログラムの良し悪し

プログラミングを学び始めたばかりの学生たちを見て気付いたのだが、彼らには、識別子の名称（変数名・関数名）やインデント（字下げ）の扱いを蔑ろにする傾向がある。 口頭で注意するに留まらず、倣うべき実例を示すことで悪癖を正してもらおうとするのだが、その指導の意図を伝えきれないことが多い。 「言葉」で示されても、あまり見慣れていない「プログラム」で示されても、初学者は自分の習慣の悪さを認識できないようだ。 ある一定以上の長さを持つプログラムを作成し、その後もたまに修正を加えるなどという経験をすると、識別子やインデントに気を付けるようになるかもしれないが、如何せん指導の負担が大きい。 より直感的に「これは悪い習慣なのか!!」と痛感してもらえるような指導方法はないものか、と考えていた頃に、可食化の応用を思いついたのだ。 初学者のプログラムの良し悪しを食べ物で表現して、直感的にその良し悪しを伝えようとするものである。

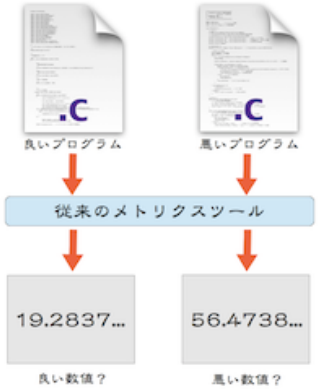


図2-1 従来のメトリクス結果提示の例

図2-2 可食化によるメトリクス結果提示の例

なお、例題を採択した経緯は「はじめに - 三つの興味（教育に関すること）」に記してある。

プログラム

- プログラムの性質と良し悪し

  そもそも「プログラムの良し悪し」とは何か。様々な文献から拾い集めてきた「良いプログラム」の例をここに示そう。

  > *実行効率の良いプログラム　移植性（ポータビリティ）の良いプログラム　見通しの良いプログラム　使い勝手の良いプログラム　反応の良いプログラム　依存性が少なく良いプログラム　可読性の良いプログラム　保守性の高い良いプログラム　気持ちの良いプログラム　…*

  本来は図2-2に示したように、良いプログラムは良いスパゲッティに、悪いプログラムは悪いスパゲッティに、という単純な対応付けを実現したかった。 しかし、示した通り「良いプログラム」の意味も様々であるから、その良し悪しを単純に対応付けたり一概に議論したりすることはできない。 それ故に、「プログラミング初学者が書くプログラムの」と断っておき、取り扱う領域（ドメイン）を限定しておいたのである。

  また、前項で漠然と「プログラミング初学者が書くプログラムの良し悪し」と述べたが、具体的には、プログラムのどのような性質に注目すれば、その良し悪しを判断できるだろうか。 プログラムの性質（指標）の種類も、良し悪しの意味と同じように多様である。 本研究の簡易調査で把握した指標について付録資料[1]を付しておくが、やはりその数は多い。

- 初学者プログラムのための指標

  さて、このような指標群を参考にしながら「初学者のプログラムに見られる特徴をよく表す指標」を探し出し、以下のようにまとめた。

  - 関数あたりの行数

    「構造化プログラミング」や「段階的詳細化」と言われるように、ある機能を実装する際に、その機能をより小さな機能の群によって実現するという考え方がある。 具体的には、ある特定の関数内のプログラム行数が多くなり過ぎないように、いくつかの関数に分解することで、プログラムの可読性（読みやすさ）を高めることに相当しよう。 しかし、彼ら初学者たちのプログラムは、全ての処理を一つの関数のみで記述していたりする。ものによっては一関数で千行を超える例も見受けられた。

  - 識別子の名称の長さ

    変数名や関数名などの識別子に適切な名前を用いておくと、後々にプログラムを見なおした時や、他人にプログラムを見てもらう際に、その内容の理解を助ける大切なヒントとなる。 しかし、彼らのプログラムには不適切な名前が散見される。特に、極度に短い名称を用いる例は多く、そのプログラムを書き下ろした本人でさえも、その変数や関数の役割を説明できないほどである。

  - インデントのずれ具合

    インデント（字下げ）を付けることで、プログラムの構造を把握しやすくなるものだが、彼らは全体の構造を気にする余裕がなく、インデントに注意を払わないようだ。 ただ、誰かに教示されて写した部分については、教示した者によるインデントが施されていたりする。 結果的に彼らのプログラムは、インデントの施し方における一貫性の欠如（インデントのずれ）が見られるのだ。

  - コメントの記述量

    識別子の名称と同様にプログラムの理解を助けるものとしてコメントが挙げられよう。 適切な名称の識別子を用いたプログラムであれば、コメントがなくとも理解に苦しむことはないだろうが、初学者向けのプログラミング演習科目ではコメントの記述を推奨している場合が多い。 それらの科目に倣い、プログラムの理解を助ける保険的な意味合いを込めて、コメント記述量を取り扱うことにする。

  ここまで説明した四つの指標を用いて、初学者のプログラムの良し悪しを判定することにする。 なお、誤解を招いてはならないので弁明しておくが、これらの悪い特徴が初学者全員に見られるわけではない。

- スパゲッティプログラム

  序論でも何度か触れている通り、プログラムを「スパゲッティ」として写像することにしたのだが、その経緯に触れる。

  唐突だが「スパゲッティプログラム」という言葉を紹介したい。 「構造化プログラミング」が盛んに叫ばれるよりも前、ソフトウェアは職人技（素人では簡単に為し得ない複雑なプログラミング）によって実現されていた。 しかし、時代の流れとともにソフトウェア開発の規模も膨張し、職人技で記述されていたプログラムは、もはや手のつけようのないものと化していた。 その「複雑極まりないプログラム」つまり保守性が著しく欠けたプログラムのことを「スパゲッティプログラム」と呼び習わすのである。

  さて、関連研究[1]では、ソフトウェアプログラムを「都市」というメタファで表現したが、それに代わる、さらに直感的なメタファはないものかと疑問を感じていた。 「都市」から最初に連想したのは「道路」であったが、同じく建造物であり面白みに欠ける。少し飛躍を試みようとして、たまたまお腹を空かしていたことを理由に「食べ物」を連想。 その日の午前中に購入した「納豆」を思い出したが、癖もあり、人によって好みが分かれるものでもあるから、より人気のあるものはないだろうか、と。 そうして思い至ったのが「スパゲッティ」である。 事前に「スパゲッティプログラム」という言葉を知っていたこともあり、この着想を得た瞬間に「これだ!!」と感じたのだった。

図2-3 「都市」から「スパゲッティ」への連想経緯

スパゲッティ

- スパゲッティの世界

  スパゲッティの織り成す広大な世界をご存知だろうか。 研究を開始した当初の私は、この単純な食材なら研究で取り扱うことも容易かろうと考えていたのだが、それは見当違いであった。 歴史・地域性・製法・調理法など、この一節では語りきれないほど多彩な様相を見せてくれる食材なのである。 本項では、その調査のほんの一部のみを垣間見ていただこう。（各項目は参考文献[3][4][5]に得たものである。）

  - 歴史

    スパゲッティの起源には諸説があり、未だに明らかにされていないが、いずれの説も、とうもろこし粉や豆粉などを水で練って食料とする文化が存在したと説明している。 最初は主婦・料理人による手作りに始まり、やがて職人が生まれ、人力の圧力機も登場。さらに産業革命とともに蒸気機関による製造自動化が実現され、さらに電動機に遷って全工程を自動化し、今に至る。

  - 地域性

    イタリア国土は南北に伸びる形をしており、南北でその気候が異なるのだが、北イタリアの気候では原料のデュラム小麦を栽培できない。代わりに軟質小麦に卵を加える事で生地を作るため、南北でスパゲッティの特徴も異なってくる。 （ただし、乾燥パスタが普及した現在はその限りではない。） また各地方の近隣諸国に影響を受けたソースがあり、そのソースと合わせたスパゲッティ料理自体やその名称にも地域性が見られる。

  - 製法

    金属製の型に生地を押し込むことでスパゲッティを作り出すのだが、その金属の種類（ブロンズ製かテフロン製か）によってもスパゲッティの性質（表面の凹凸具合）が変化する。 また、原料はデュラム小麦100%とし、色付けする場合には自然食品のみ用いることができる、とイタリアにおけるパスタの法律に記載されていたり。

  加えて、スパゲッティに関するアンケート調査や、ゆで時間や待ち時間に関する調理実験を行い、スパゲッティにまつわる知見を蓄積していった。 また、研究を始めた頃から、機会を見つけては積極的にスパゲッティを食べるようにしていた。 付録資料[2]に掲載する通り、店内で15種、他の調理済みのものを7種、自主調理を16種、計38種類を食し、さらに調理実験や研究室内での自炊も重なり、スパゲッティは私の主食になってしまった。 もはや完全な蛇足だが、今ではお気に入りの銘柄（バリラ No.5）を5kg単位で購入するようになり、今日も研究室にてアル・デンテの悦びを噛み締めている。

図2-4 実験で24時間放置したスパゲッティ

図2-5 スパゲッティ・コレクション

図2-6 バリラ No.5 1.7mm（5kg）

- スパゲッティの性質と良し悪し

  一言に表現する「スパゲッティ」だが、その姿が多様であるということをお分かりいただけたのではないだろうか。 そして、プログラムについての議論と同じように、スパゲッティの良し悪しについてもまた様々な解釈があるものだ。 コストパフォーマンスの良さ、保存性能の良さ、味の良さなどなど。スパゲッティに関する調査で洗い出した、それらの指標群（一部抜粋）を次に示そう。

  > - 品質（銘柄、材料、製造法、etc.）
  > - ゆで時間、作り置き時間
  > - 食感、コシ、歯応え、弾力感
  > - 太さ、長さ、形状
  > - 値段、料理名
  > - 風味、香り、味、塩加減
  > - もつれ具合
  > - 乾きやすさ（オリーブオイル量）
  > - ...

写像

ここまで、本研究における論旨の構成要素であるプログラムとスパゲッティについて述べてきた。 最後に、それらをどのように写像するか（対応付けるか）について述べ、また、写像を実現する仕組みについても説明し、 序論 - メトリクスの新しい枠組みの提案で述べた主張の論拠を示す。

- 指標同士の対応付け

  上で示した「プログラムの性質」及び「スパゲッティの性質」の知見に基づき、それらの指標同士の対応付けについて以下の通り提案する。これにより、（初学者としての）良いプログラムは良いスパゲッティに、悪いプログラムは悪いスパゲッティに写像することができるはずである。

プログラムの指標	スパゲッティの指標
関数あたりの行数	作り置き時間
識別子の名称の長さ	スパゲッティ料理の名称
インデントのずれ具合	ゆで時間のばらつき
コメントの記述量	オリーブオイル量

  - 関数あたりの行数 → 作り置き時間

    構造化を蔑ろにして一つの関数内に膨大なプログラムを記述することがあるのだが、この場合「関数あたりの行数」は大きくなる。 そのような「悪いプログラム」の場合は、作り置き時間を長くし、水分が蒸発して乾燥した「悪いスパゲッティ」として表現する。 ちょうど、「プログラムのまとまり」を「スパゲッティのまとまり」として上手く喩えられているはずである。 逆に、適切に構造化されたプログラム、つまり「良いプログラム」の場合は、作り置き時間を短くし、できたての「良いスパゲッティ」として表現する。

  - 識別子の名称の長さ → スパゲッティ料理の名称

    変数名や関数名の平均長を算出し、それを「識別子の名称の長さ」として扱う。 その長さが極端に短いまたは長い「悪いプログラム」の場合、スパゲッティ料理の名称も極端に短くまたは長くし、「悪いスパゲッティ」として表現する。 適切な長さの「良いプログラム」の場合、料理名も適切な長さの分かりやすいものにし、「良いスパゲッティ」として表現する。 これは単なる名称であり、食感に対して直接影響を及ぼすものではないが、名称はその料理の印象を与える意味で重要な情報であり、 間接的に食感への影響があると捉え、この指標を採用した。

  - インデントのずれ具合 → ゆで時間のばらつき

    インデントの付け方に一貫性がない「悪いプログラム」は、ゆでの浅い麺とゆで過ぎの麺とが混同したような、ゆで時間のばらつきがある「悪いスパゲッティ」として表現する。 逆に一貫性がある「良いプログラム」は、ゆで時間も統一された「良いプログラム」として表現する。

  - コメントの記述量 → オリーブオイル量

    ゆで上げたスパゲッティを放置しておくと、水分が蒸発して乾燥してしまうが、その乾燥を防ぐためにバターやオリーブオイルを絡ませることがある。 コメントが、プログラムに込められた意味を逃がさない（忘れない）ようにするための備忘録だと考えれば、水分を逃さないためのオリーブオイルに相当しよう。 コメントがない場合は、オリーブオイルを一切使わないため水分の蒸発しやすいスパゲッティ、 適度にコメントが記述されているプログラムの場合は、オリーブオイルを適量絡めたスパゲッティ、 コメントが多すぎる場合は、プログラムを読む妨げになると考え、オリーブオイル漬けのスパゲッティとして表現する。 また、オリーブオイル量という指標は、一つ目の指標「作り置き時間」に干渉するものである。 多少作り置き時間が長くとも、オリーブオイルが絡まっておれば乾燥を遅らせることができてしまうのだ。 ただ、プログラムの悪い点をコメントの記述によって補っていると捉えれば、むしろ都合の良い干渉なのかもしれない。 （採用の成否は実験によって示すことにしよう。）

- 写像という構造

  次に、この対応付けを実現するための構造について述べる。

  ソフトウェアメトリクスの研究に取り組む身として、その長短を知っておくべきだろうと考え、既存のメトリクスツールを試用してみたこと

がある。 具体的には、それぞれ図1-1や図1-2として示していた、 「SourceMonitor」[6]及び「Eclipse Metrics plugin」[7]だが、 いずれもソフトウェアプログラムを入力とし、数値やグラフを出力する「仕組み」だ。また、今回開発する可食化のためのメトリクスツールは、ソフトウェアプログラムを入力とし、スパゲッティを出力する「仕組み」である。

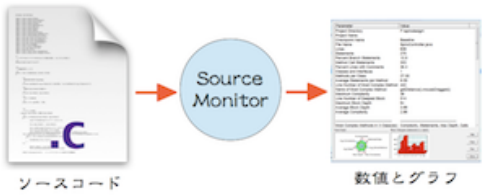


図2-7 SourceMonitorによる数値・グラフへの変換

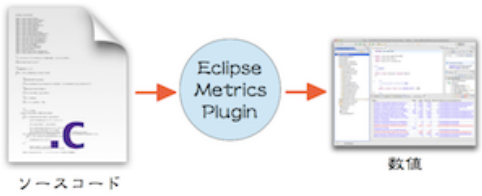


図2-8 Eclipse Metrics pluginによる数値への変換

図2-9 今回開発するメトリクスツールによるスパゲッティへの変換

先に示した既存の二例と同じようにして「可食化のツール」を開発するだけで良いのだが、果たして本当にそれで良いだろうか。 何かあるものを入力として別のあるものを出力する構造、つまり、何かあるものを別のあるものに対応付けて写像する構造が、どの例にも見られるではないか。 この構造を括り出して抽象化すれば、また別のメトリクスツールを開発する際の便（再利用性）を高められるはずである。 ちなみにここでは、元のプログラムを「写像元」、メトリクスツールを「写像役」、スパゲッティを「写像先」と表現することにする。

図2-10 抽象化した「写像」の仕組み

一度、抽象的な話に持ち込んでおきながら、また具体的な話に戻るようで恐縮だが、ソフトウェアプログラムをスパゲッティへ写像することについて、より詳細に表した図2-11を示す。 前項（指標同士の対応付け）で示した四つの対応付けを参照しながら、図中の写像関係を見ていただきたい。

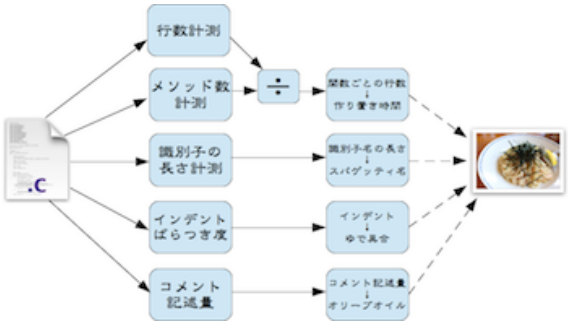


図2-11 スパゲッティへの写像（詳細）

図2-9では、まるで一枚岩のような写像役であったが、それは前項で示した通り四つの対応付けに分解できる。 また「関数あたりの行数」については、さらに三つの写像（「行数計測」「メソッド数計測」「÷」）に分解できることから、図2-11のように表現することができたのだ。 さて、この図をまた抽象化してみると、図2-12のようになる。

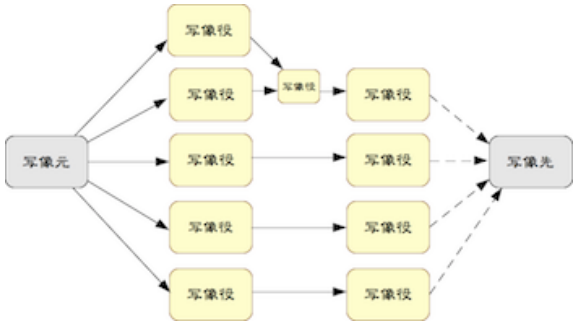


図2-12 抽象化した写像の構造

本研究で実現しようとしている「プログラムをスパゲッティに変換する」という行為が、ここで述べた抽象的な構造「写像」のみで構成できるという点をご理解いただけたであろう。 次項では、この抽象化（汎化）により議論できる「大切なこと」について述べよう。

- メトリクスグラフ

  図2-11を見ていると、ノード（写像元・写像役・写像先）とアーク（繋ぎ役）によって構成されるグラフであることに気付く。 また、その繋ぎ役には方向性があり、かつ、特定二組のノード間を結ぶアークは複数存在し得ることから、これは多重有向グラフであり、今後これを「メトリクスグラフ」と呼び表すことにする。

  大切なことは、ノードやアークといった構成要素（小さな部品）のみによって、あらゆるメトリクスの写像関係を表現できるという構造にある。 言い換えれば、その部品群をソフトウェアとして実装できさえすれば、あらゆるメトリクスツールを実現できるのだ。 最後に、この「メトリクスグラフ」こそが、提案したい枠組みであるということをここに宣言し、 以上を、序論 - メトリクスの新しい枠組みの提案で示した主張の論拠とする。

計画

上述した理論を根拠として論旨の主張を行いたい。その主張のための計画を本項に示す。

評価実験による有効性の実証

以下は、第一の論旨（「可食化」の提案）に関する実施計画である。

写像 - 指標同士の対応付けで示した写像関係に従って、プログラムをスパゲッティとして可食化し、食べる行為を通して直感的な理解が得られるということを実証したい。 実際には、被験者に試食及びアンケート評価実験を実施し、可視化と可食化の双方に対して「直感的な理解・認知」のしやすさを評価してもらう。 その結果を統計的に分析し、直感的な情報表現として可食化が有効かどうかを判定する。 この実験の詳細については、第4章 実験をご参照いただきたい。

枠組みによるメトリクスツールの実現

以下は、第二の論旨（メトリクスの新しい枠組みの提案）に関する実施計画である。

理論で示した枠組み「メトリクスグラフ」を援用した新しいメトリクスツールを開発し、当初の主張である「あらゆるメトリクスツールを実現できる」ことを実証したい。 まずは、メトリクスツールを開発するための計画として、その構成要素を明らかにし、それぞれの役割も簡単に述べておく。 ここで説明し切らない詳細については、第3章 道具と作成物をご参照いただきたい。

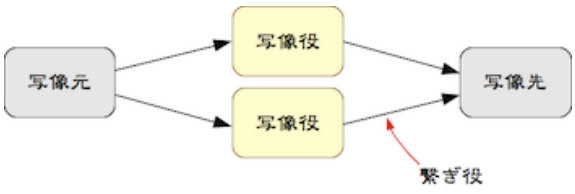


図2-13 メトリクスグラフの例

- 写像役（ノード）

  ある一定の規則に従って、入力を出力に変換するものである。数学的に言えばベクトル値関数（多入力・多出力を許す関数）に相当する。例えば、「3」と「4」を入力して「加算」という規則に従い「7」を出力するものは、この写像役にあたる。

- 写像元（ノード）

  写像役に「入力するもの」がこれに相当する。つまり、メトリクスの計測対象となるものである。例として、人体やソフトウェアプログラムなどが挙げられよう。

- 写像先（ノード）

  写像役から「出力されるもの」がこれに相当する。つまり、メトリクスの計測結果を意味するものである。例として、ログファイルやスパゲッティなどが挙げられよう。

- 繋ぎ役（アーク）

  上記三種類のノードを接続するためのものである。数学的概念で言えば関数の合成に似ている。具体的には、あるノードの出力値をあるノードの入力値として引き渡す役割を担う。

学生証番号：947424　氏名：宮崎 雅文

第3章 道具と作成物

本章では、論旨の主張のために開発する作成物（新しいメトリクスツール）と、その開発に必要となる道具（概念・環境・技術要素）について述べる。

開発の準備

要求

これまでに述べた論旨二点の主張を目的とするソフトウェアの開発を行う。 「可食化の実現」を本願としながら、「枠組みの利用」により再利用性の高いメトリクスツールを開発することが目標となる。 従って、枠組み「メトリクスグラフ」の構成要素である「ノード」や「アーク」に相当する仕組みを、それぞれ可能な限り独立した機能として実現したい。 また最終的には、ソフトウェアプログラムからスパゲッティを得られるように、ツールを仕上げなければならない。

開発計画

本開発では、大きく分けて二つの機能を実現しなければならない。 また、それらは独立しているものの、運用時には共に協調して動作するものであるから、一方の設計に大きな変更があった場合には、もう一方にも影響を及ぼす可能性がある。 そのリスクを吸収するというねらいから、アジャイルソフトウェア開発の一であるXP（エクストリームプログラミング）の考え方を参考に開発を行う。 （cf. 参考文献[5]）

2012年10月25日から本開発を始動し、アイデアの捻出や反復開発を経て、要求事項の達成を年内（2012年12月末日まで）に遂げ、開発を収束させるものとする。

アイデア

上記のメトリクスツールを開発するにあたり、その実現に向けて二つのアイデア（初期計画）を用意していた。それぞれを示しておこう。

- ビルドツールmakeを転用した「メトリクスプラグイン」 —— 写像役（ノード）の実現

  入力を出力に変換する関数のようなもの（写像役。以降「メトリクスプラグイン」とも呼ぶ。）を用意しなければならないが、それなら、その関数を何かしらのプログラミング言語で実装すれば実現できよう。 しかし、「再利用性の高いツール」というコンセプトを考えると、特定のプログラミング言語に媚びることは芳しくない。 誰でも好きな言語を用いて、独自のメトリクスプラグインを開発できるようにしておきたいのだ。 ここで着想したのが、ソフトウェアプログラムのビルドツールとして広く用いられている「make」を転用することである。

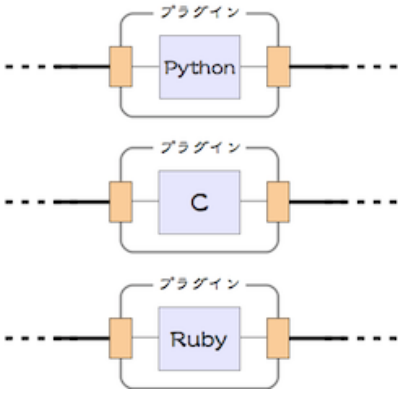


図3-1 makeを用いるプラグイン

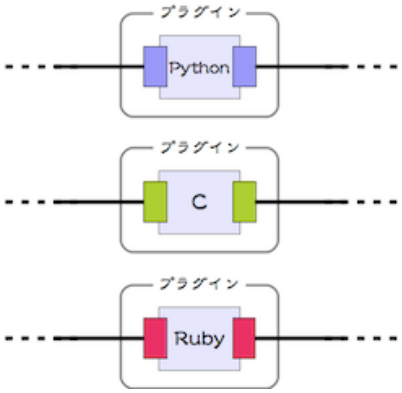


図3-2 makeを用いないプラグイン

「make」を利用しない場合は、プラグインの機能実装に使われる言語の違いをアーク側（繋ぎ役）が考慮しなければならないが、それではノードとアーク間に依存が生じ、独立した機能として実現できない。 そこで「make」を利用することにより、その言語の違いをプラグイン側で吸収すれば良いのだ。 つまり、メトリクスプラグインを利用するためのインタフェースとして「make」を転用するのである。

メリットはこれに留まらない。プラグインと繋ぎ役との間で共通の界面を利用できることはもちろん、プラグインと人間との間においても同じ効用が期待できるであろう。 それぞれの実装言語の使い方を知らずとも、ビルドツール「make」の扱いさえ理解しておれば簡単に扱えるのだ。 また、もはや「人間」に限定する必要もなく、プラグインと何か他のシステムとを連携させる場合も同様であるから、この統一されたインタフェースによる利益は大きいと言える。 これなら、再利用性に関するコンセプトを充分に満たしているはずだ。

なお、「メトリクスプラグイン開発ガイドライン」を付録資料[13]として添えておく。

- メトリクスグラフを表現するための言語「SynapseScript」 —— 繋ぎ役（アーク）と写像元・写像先（ノード）の実現

  メトリクスプラグイン（写像役）や写像元・写像先といった「ノード」を繋ぎ合わせることで、メトリクスグラフを表現することができるが、その繋ぎ合わせ（接続関係）を表現する方法として「SynapseScript」という言語を提案したい。 その紹介に先立ち、グラフに関する二つの議論を済ませておきたい。まずは多重グラフの表現方法について述べよう。

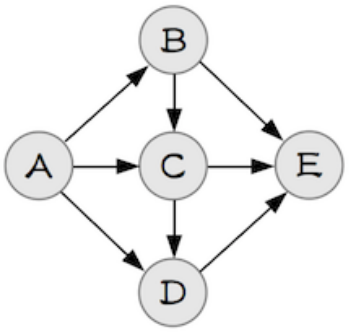


図3-3 グラフの例

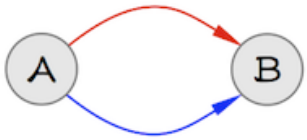


図3-4 多重グラフの例

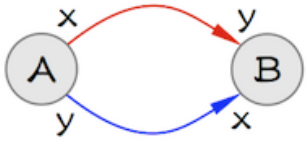


図3-5 ラベル付き多重グラフの例

本来のグラフにおいて、ノードの接続関係を表現するには、接続したい二つのノードを一組として表現するだけで良い。その表現そのものがアークを表すのだ。 例えば、図3-3のグラフにおけるノードの接続関係を表すと、次のようになる。

```
(A,B) (A,C) (A,D) (B,C) (B,E) (C,D) (C,E) (D,E)
```

丸括弧やカンマに意味はない。（有向グラフであるから、括弧内のノードの順序には意味があるのだが、）大切なのは、二つのノードが一組として表されている、その構造である。 次に、図3-4のような多重グラフであればどうであろう。ある二つのノード間が、それぞれ別の二本のアークで接続されているなら、どのように表現すれば良いのか。

```
(A,B) (A,B)
```

これは失敗である。二つのアークを表現してはいるが、それらを区別することができない。 それなら、アークの両端点（ノードとの接続点）に対して識別可能なラベルを付した図3-5は、表現できるだろうか。

```
(A.x,B.y) (A.y,B.x)
```

あるノードAに付したラベルxは、ピリオドを挟んで「A.x」とすることで容易に表現できた。

第2章 理論と実施計画 - 理論（写像 メトリクスグラフ）で述べたとおり、メトリクスグラフもまた多重グラフ（多重有向グラフ）であるから、 ラベルが必要になる場合がある。その簡単な例として、除算（割り算）のメトリクスグラフを図3-6に示す。

図3-6 除算を表現するメトリクスグラフ

除算を行うメトリクスプラグインを中心に、写像元（入力値）二つと写像先（出力値）三つが描かれている。 このように、多入力または多出力を持つノードは、各接続点にラベルを付けて識別可能にし、接続関係を表現できるようにしなければならない。

さて、残る一つは必ずしも議論を要することではないのだが、接続関係を表現する際の冗長性（無駄）を省くために考えた工夫があるので、ここで述べておきたい。

図3-7 連続した接続を有するメトリクスグラフ

図3-7のグラフについて、その接続関係を素直に書き下ろせば、次のようになる。

```
(A,B) (B,C) (C,D)
```

連続する接続を表現したものであるが、一つの接続ごとに改めてノードを記さねばならず、やや冗長さを含む。これを次のように書き直せば、どうであろう。

```
(A,B,C,D)
```

重複した記述、つまり冗長性を排除できたことは自明である。 もちろん、その接続の意味合いによっては、分けて書くべき場合もあろう。大事なことは、この接続関係を記述する者に対して、表記の自由を与えることである。 一方を無闇に排除したりすべきではなく、どちらの表記も許容することが肝要である。表記の細事ついて考えを凝らすべきは設計者ではなく記述者なのだ。

これら二つの議論を踏まえ、メトリクスグラフを表現することに適した言語を設計・開発する。 わざわざ新しい言語を開発する必要はないだろうが、それでも言語開発に拘ったことには理由がある。 詳しくは「はじめに - 三つの興味（言語処理系に関すること）」を参照されたい。 それから、言語「SynapseScript」に関する詳細は「SynapseScript 言語マニュアル」をご覧頂きたい。（cf. 付録資料[1]）

制限事項

これまで触れなかったが、本開発にあたりその制限事項について簡単に付け足しておく。

- 「Cのプログラム」をメトリクス対象とする
  初学者のプログラムの良し悪しを、と議論してきたが、実を言えば、そのいずれもC言語のプログラムを念頭に置いて展開してきた。 本来は、言語の違いも吸収してスパゲッティを出力したかったのだが、解析の方法を一意に定めるため具体的な言語を決めておくことにする。

- スパゲッティへの写像は「レシピの出力」で実現する
  プログラムを入力してスパゲッティを出力する、と述べてきたが、メトリクスツールで写像した結果をいきなり現実空間のスパゲッティに変換できるわけではない。 結果の数値を基に調理して初めて、目当てのスパゲッティを得ることができる。 つまり、今回開発するメトリクスツ

ールは、「スパゲッティ」を出力するわけではなく「スパゲッティのレシピ」を出力するものである、と断りを入れておく。

- 言語処理系の実装にプログラミング言語「Smalltalk」を用いる

  SynapseScriptの言語処理系を開発しなければならないが、その実装を行う言語として「Smalltalk」を、またその処理系として「VisualWorks 7.8」用いる。 研究室にて普段からよく利用していたこと、理不尽な仕様が比較的少ないこと、プログラム走行時のテストが非常に容易であること、などを理由として採択に至った。

反復開発

開発計画の通り短期的な開発の反復を基本として、また上記アイデアを基盤として、要求を充足する機能（ソフトウェア）の開発に取り組む。

第1期：言語設計とプラグイン設計

- 言語設計

  開発する言語「SynapseScript」は、メトリクスグラフを表現するためのものであるが、可能な限り簡単な記法かつ読みやすい記法を採用したい。 実際に記法の具体案を十数例ほど用意し、吟味の上で一つに絞り込んだ。以下に例題グラフ（図3-8）と、そのグラフに対応するSynapseScriptの記述例を示そう。

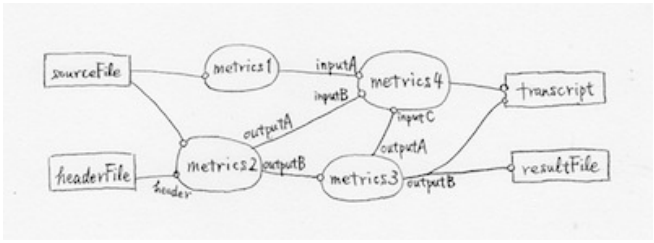


図3-8 サンプルのメトリクスグラフ

```
sourceFile := File("/path/to/SourceFile.c").
headerFile := File("/path/to/HeaderFile.c").
metrics1 := Metrics("/path/to/Metrics1").
metrics2 := Metrics("/path/to/Metrics2").
metrics3 := Metrics("/path/to/Metrics3").
metrics4 := Metrics("/path/to/Metrics4").
transcript := Transcript().
resultFile := File("/path/to/ResultFile.txt").

sourceFile
    to: metrics1
    to: metrics4.inputA
    to: transcript.
sourceFile
    to: metrics2.source
    send: outputA to: metrics4.inputB.
headerFile
    to: metrics2.header
    send: outputB to: metrics3
    send: outputA to: metrics4.inputC.
metrics3.outputB
    to: transcript.
metrics3.outputB
    to: resultFile.
```

SynapseScriptは「ノードの定義文」と「アークの定義文」の二種類により構成される。示した例では、上部にノードの定義を、下部にアークの定義を記述している。 1行目は写像元となるノードを定義して、それ以後は、sourceFileという名称で扱うことを宣言している。 3行目は写像役（メトリクスプラグイン）となるノードを、8行目は写像先となるノードを定義して、それぞれの別名称を宣言している。10-13行目は、四つのノードを連続に接続することを表している。 sourceFileをmetrics1に入力し、その写像結果（出力）をmetrics4のラベルinputAに入力し、さらにその結果をtranscriptに入力している。

このように記法が定まったことから、字句解析用の正規表現と構文解析用のBNF（バッカス・ナウア記法）を用意した。これは言語マニュアルに添えてある。（cf. 付録資料[4]）

- プラグイン設計

  さらにプラグインの設計を行い、これまで漠然としていた実現方法について具体的に検討した。 一つのディレクトリを一つのプラグインと見立て、そのディレクトリ内に具体的な写像の仕組みを格納し、ビルドツール「make」を経由して写像処理を実行できるようにする。 ここ

で写像の仕組みとは、makeの動作規則を定義するテキストファイル「Makefile」ならびに、具体的な写像処理を記述したプログラムのことである。 また、プラグインは幾つかの機能を実現しなければならないが、その方法としてmakeのターゲットを活用することにした。 makeは、端末上で「make」と入力すると動作するのだが、後部にターゲット名と呼ばれるものを付すことで実行する動作規則を変更することができるため、これを機能選択の仕組みとして利用するのだ。

それから、プラグインに入力値を引き渡す方法として、makeのマクロ定義を用いる。これが変数のような役割を果たすことで、プラグインへの値の入力を可能にする。 また、プラグインの写像結果の出力形式として、JSONを用いる。単純な記法で連想配列（名前と値の一組を要素とする配列）を表現できることから、ラベル付きの複数値を出力するメトリクスプラグインに適していると考えた。

以上の通り、SynapseScriptの言語設計とメトリクスプラグインの仕様設計を行い、実現に必要となる技術項目を洗い出すことができた。 ただし、この時点での言語設計及びプラグイン設計は変更される可能性があるということに注意されたい。（実際この後、仕様を変更することになる。） なお、第1期開発は、2012年10月25日から同年11月14日まで実施した。

第2期：字句解析と構文解析

言語の定義を終えたため、その言語処理系の実装を行う。 SynapseScriptで記述された表現を解析し、その構文木を得ることが目標となるが、これは既存の字句解析器生成系（lex/flex）や構文解析器生成系（yacc/bison）を用いることで容易に実現できる。 しかし、「はじめに - 三つの興味（言語処理系に関すること）」で述べた通り、処理系そのものにも興味を抱いていたことから、 字句解析器と構文解析器を独自に実装することにした。

- 字句解析

  第1期開発で用意した正規表現に基づいて図3-9に示すオートマトンを描き、字句解析を実装した。

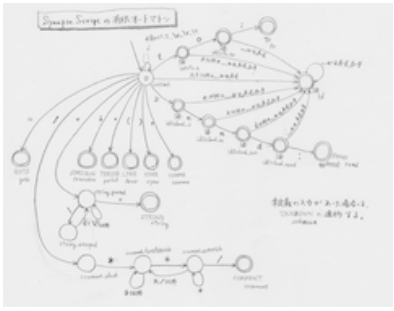


図3-9 字句解析のためのオートマトン

- 構文解析

  構文解析の方法には幾つかの種類があるが、中でも以前から学びたいと考えていたLR構文解析（SLR(1)）を実装した。

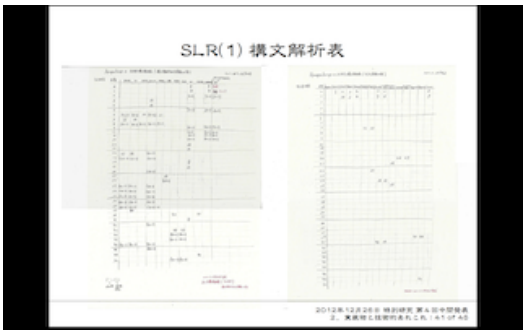


図3-10 SLR構文解析表（研究中間発表(4)より）

- テスト

  実装した解析器の動作を検証するため、次のサンプルプログラムを解析し、その結果（構文木）を図示することにした。その様子を図3-11に示す。 なお、このサンプルプログラムが表すメトリクスグラフの意味を考える必要はない。

```
transcript = Transcript("hoge");
transcript send: output to: transcript.input;
```

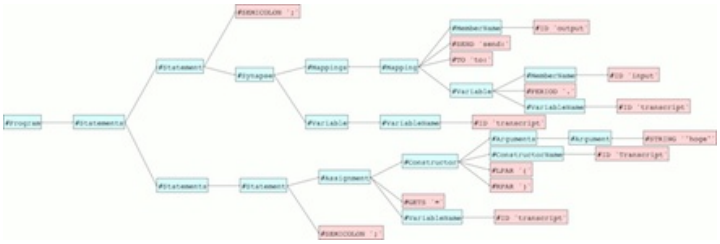


図3-11 サンプルプログラムの構文木

以上の通り、SynapseScriptの字句解析及び構文解析を実現することができた。 この開発に際して参考文献[10] [11]が大変役立った。 なお、第2期開発は、2012年11月14日から同年12月1日まで実施した。

第3期：Smalltalkからシェルの利用

Smalltalk上でSynapseScriptを解析し、得られたメトリクスグラフの写像関係に従ってメトリクスプラグインを利用することになる。つまり、SmalltalkからmakeコマンドⅠシェルⅠを利用できなければならない。 その仕組みは標準で提供されてはいるのだが、不具合や不都合（マルチバイト文字の問題）が存在するため利用できない。そこで別の手段として、SmalltalkからCを経由してシェルを利用することにする。

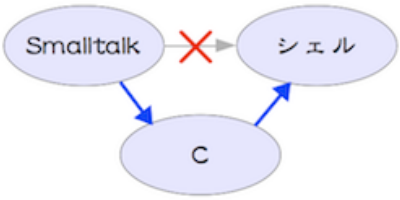


図3-12 Smalltalkからのシェルの利用

- Smalltalk → C

  これはVisualWorksに付属しているDLLCCという仕組みにより、容易に実現することができる。なお、DLLCCについては参考文献[12]に詳しい。

- C → シェル

  Cの標準ライブラリに用意されている関数systemで、シェルのコマンドを実行することができるが、その実行結果（出力）を得られないという欠点がある。 そのため別の手段として、fork、pipe、dup2、execといったシステムコールを利用し、Cからシェルの子プロセスを立ち上げてパイプで連携することによって一連のシェルコマンドを実行することができた。

- Smalltalk → シェル

  上記二項を合わせ「ShellInterface」及び「ShellInterpreter」と称して、Smalltalkからシェルの利用を実現した。

- 問題点

  開発したソフトウェアはバグを含む。シェルの子プロセスの管理またはパイプの管理に不備があるようで、結果が応答されず、処理も終了しないことがある。 また、子プロセスに対するEOF（End Of File）の送信を実現できなかったため、EOFを受けるまで待機するコマンドを実行すると、そのコマンドの実行を止めることができない。

ひとまず、Smalltalkからのシェルの利用を実現することはできた。残されている種々の問題については、今後の開発で改善してゆくことが期待される。 なお、第3期開発は、2012年12月2日から同月12日まで実施した。

第4期：インタプリタ

第2期開発の成果物を用いてSynapseScriptによる記述を解析できるが、そこで得た構文木に従ってメトリクスグラフを構築し、実際の写像処理を実行できるインタプリタを開発したい。 その開発にあたり、「SynapseScript」という名称の由来でもある「神経系」の構造を模して設計することにした。概念図を図3-13として示す。

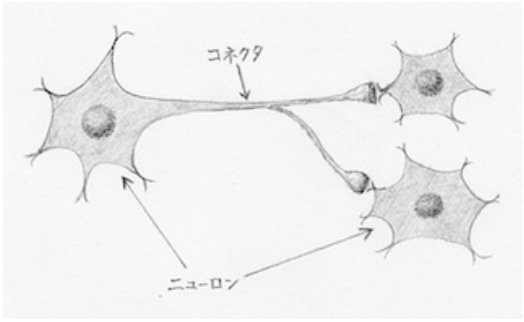


図3-13 神経系を模した概念図

ノードをニューロン（神経細胞）に喩えると、その神経線維（軸索と樹状突起）がちょうど繋ぎ役（コネクタ）となる。 ニューロンが神経線維を通して、他のニューロンに対して物質を伝達するように、ノードもコネクタを通して、他のノードに情報を伝達すれば良い。 伝達機能を持つオブジェクトを拵えた上で、構文木から神経構造（メトリクスグラフ）を構築すれば、再帰的に全ての写像を実行することができよう。

この考えに基づいて「SynapseInterpreter」という名称でインタプリタを開発した。 動作テストとして、簡単なメトリクスプラグインを作成し、それを用いるメトリクスグラフを記述・実行したところ、確かに設計通りに動作することが確認された。 なお、第4期開発は、2012年12月15日から同月20日まで実施した。

第5期：プラグイン仕様の策定と再考

第4期開発の動作テストでは、メトリクスプラグインの試作を作り利用していた。 その際も特段の問題は起こらなかったため、メトリクスプラグインの開発方法を具体的に定めて、ガイドラインとして策定することにした。

ところが第6期開発に入って仕様の問題が発覚した。 メトリクスプラグインにデータを入力する際、makeコマンドにおけるマクロ定義として記述すると先に説明したが、入力データに特殊文字が存在する場合にエスケープ処理を施さねばならない。 また、そのエスケープはデータの受け渡しを行うごとに必要とされるため、「エスケープしたものをエスケープし更にエスケープする」といった複雑な処理を行う羽目になるのだ。

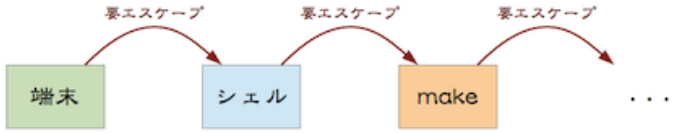


図3-14 幾段にも重なるエスケープ処理

そして、そのエスケープの施し方は環境により異なるため、特定の環境で正しく動作しても、別の環境で動作するとは限らない。これでは、再利用性を高くするというコンセプトにも反しよう。 統一的なエスケープ処理で済ませるために試行錯誤を重ねたのだが、具体的な方法を見出だせなかったため、その代案として、エンコード規格の一つ「Base64」を援用することにした。 どのようなデータも、Base64でエンコードすることで「エスケープを必要としない文字列」に変換できるため、データの受け渡しが幾段に重なろうとも問題は起こらないのである。

本来ならデータをそのままの状態で扱い、その可読性を維持したかったのだが、重なるエスケープに頭を悩ませることこそ本質的ではないため、止むを得ず妥協に踏み切った。 ともあれ、ある種のメタ系（文字列で文字列を取り扱おうとする構造）の難しさを垣間見ることができ、学びとなった。 なお、第5期開発のうち、特に仕様策定は2012年12月21日のみ、また仕様再考とそれに伴う追加実装は同月29日から翌年2013年1月1日まで実施した。

第6期：可食化に向けたプラグイン設計と開発

ソフトウェアプログラムをスパゲッティとして可食化するために、その写像を表現したメトリクスグラフ（図3-15）を用意した。

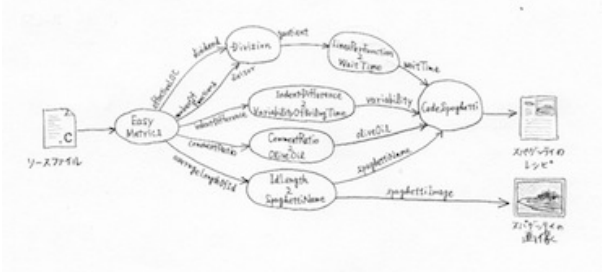


図3-15 スパゲッティへの可食化を表現したメトリクスグラフ

また、このグラフを構成している各メトリクスプラグインについて、その実現方法を説明する。 ちなみにメトリクスプラグインは、開発者の好きな言語を利用することができる。そのことを示すため、以下に挙げるプラグインはそれぞれ異なる言語で実装している。

- EasyMetrics

  Cで記述されたプログラムを入力し、基本的な指標を得るためのメトリクスプラグイン。得られる指標値は以下の通り。

種別	ラベル名	説明
入力	input	解析対象のCプログラム。
出力	linesOfCode	Lines of Code。プログラムの行数。effectiveLOCの値をそのまま採用する。
出力	effectiveLOC	eLOCとも。空白行・コメント行・括弧のみの行を除いた行数。
出力	physicalLOC	pLOCまたは物理LOCとも。単純な意味のプログラム行数。コメント行なども全て含む。
出力	logicalLOC	lLOCまたは論理LOCとも。プログラムの構成単位「文」の数を意味する。
出力	commentLines	コメント行数。
出力	numberOfFunctions	関数の宣言数。
出力	averageLengthOfId	識別子（変数名や関数名）の長さの平均。単位は文字。
出力	indentDifference	一行あたりのインデントのずれ具合。単位は文字。
出力	standardIndent	標準インデント量。そのプログラムで最も多く使われているインデント量。
出力	commentRatio	コメント記述の割り合い。commentLines÷physicalLOCによって求める。

  解析のために字句解析器・構文解析器が必要となるが、これにはflex及びyaccを利用して生成している。

  以下は、「標準インデント量」と「インデントのずれ具合」の定義について述べる。 通常のインデントで半角スペースを四つ用いる場合、その「4」という数値が「標準インデント量」に相当する。 また「インデントのずれ具合」の説明するため、次の二図をご覧頂きたい。

```
int main(void) {
    int inputValue;
    scanf (“%d”, &inputValue);
    if (inputValue >= 60) {
        printf (“Good!!\n”);
    } else {
        printf (“Too bad...\n”);
    }
    return EXIT_SUCCESS;
}
```

図3-16 インデントのずれ（ばらつき）がないプログラム

```
int main(void) {
  int inputValue;
    scanf (“%d”, &inputValue);
    if (inputValue >= 60) {
    printf (“Good!!\n”);
    } else {
    printf (“Too bad...\n”);
}
return EXIT_SUCCESS;
}
```

図3-17 インデントのずれ（ばらつき）があるプログラム

  図3-16及び図3-17に同じ内容のプログラムを示したが、それぞれインデントの整え方が異なる。 標準インデント量が「4」である場合、図中橙色の補助線に沿ってプログラムを記述するべきだが、右のプログラムはその補助線に沿っていない。 その補助線からずれている文字の数を全て数え上げ、単位行あたりの数値としたものが「インデントのずれ具合」である。

- Division

  除算（割り算）を計算するメトリクスプラグイン。2入力3出力。中核となる実装言語はPerlである。

種別	ラベル名	説明
入力	dividend	被除数。割られる数。
入力	divisor	除数。割る数。
出力	quotient	商。
出力	floorQuotient	端数を切り捨てた商。
出力	remainder	剰余。割り算の余り。

- LinesPerFunction2WaitTime（関数あたりの行数 → 作り置き時間）

  関数ごとの行数から、スパゲッティをゆで上げた後の待ち時間を算出するメトリクスプラグイン。中核となる実装言語はAWKである。

種別	ラベル名	説明
入力	input	関数あたりの行数。
出力	output	ゆで上げ後の待ち時間。単位は分。

- IdLength2SpaghettiName（識別子の名称の長さ → スパゲッティ料理の名称）

  識別子（変数名や関数名）の長さの平均から、スパゲッティ料理の名称や画像を得ることができるメトリクスプラグイン。中核となる実装言語はPHPである。

種別	ラベル名	説明
入力	input	識別子（変数名や関数名）の長さの平均。単位は文字。
出力	spaghettiName	スパゲッティ料理の名称。
出力	spaghettiImage	名称に対応したスパゲッティ料理の画像のファイル名。絶対パス。

- IndentDifference2VariabilityOfBoilingTime（インデントのずれ具合 → ゆで時間のばらつき）

  インデントのずれ具合から、スパゲッティのゆで時間のばらつきを算出するメトリクスプラグイン。中核となる実装言語はRubyである。

種別	ラベル名	説明
入力	input	一行あたりのインデントずれ具合。単位は文字。
出力	output	ゆで時間のばらつき。単位は分。

  ここで「ゆで時間のばらつき」とは、「最も早くゆでた麺」と「最も遅くゆでた麺」との時間差を言う。

- CommentRatio2OliveOil（コメントの記述量 → オリーブオイル量）

  プログラム中のコメント記述量から、乾燥防止としてのオリーブオイル量を算出するメトリクスプラグイン。中核となる実装言語はJavaである。

種別	ラベル名	説明
入力	input	コメントの記述量。単位はパーセント。
出力	output	オリーブオイル量。「少量」「適量」「大量」などと出力する。

- CodeSpaghetti

  スパゲッティの各指標値から、スパゲッティのレシピを出力するメトリクスプラグイン。中核となる実装言語はPHPである。

種別	ラベル名	説明
入力	waitTime	ゆで上げ後の待ち時間。単位は分。
入力	spaghettiName	スパゲッティ料理の名称。
入力	variability	ゆで時間のばらつき。単位は分。
入力	oliveOil	オリーブオイル量。
入力	imageFile	スパゲッティ料理の画像ファイル名。
出力	output	スパゲッティのレシピを表現するHTML。

なお、第6期開発は、2012年12月21日から翌年2013年1月6日まで実施した。

第7期：写像具合の調整

プログラムの指標値からスパゲッティの指標値を算出するメトリクスプラグインとして四種類を実装したが、まだ「入力値をどのような出力値に対応づけるか」の調整が残っている。 ここまで取り組んできた調査結果に基づき何かしらの根拠を添えて、その調整を行いたい。ただし、この後に実施する予備実験の結果によっては、その写像具合を再度調整する可能性がある。

また、実際に「良いプログラム」や「悪いプログラム」があれば対応付けの議論も楽になると考え、例題プログラムを五種類ほど用意した。（cf. 付録資料[4]） さらに、この対応付けを行うにあたり同輩に頼んで、彼らが数年前に拵えたというプログラムを提供してもらった。そのご厚意に心から感謝する。

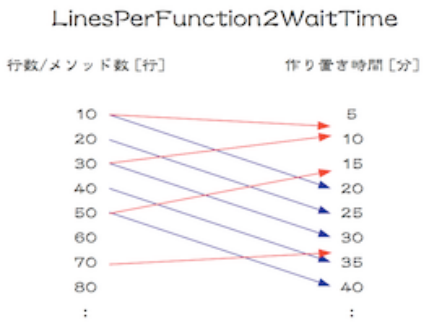


図3-18 写像具合の調整の例

- 関数あたりの行数 → 作り置き時間（LinesPerFunction2WaitTime）

　用意した例題プログラムの中に、たった一つの関数のみで実装したものがある。このプログラムの「関数あたりの行数」は「54」であった。このことを根拠に、その値が50を超えるプログラムに対して、0より大きい作り置き時間を与えることにした。 50を超える行数が多くなるほど作り置き時間を長くしたいのだが、その数値を定めようにも根拠がない。 そこで、まずは仮の数値を設定し、後の予備実験の結果を鑑みて調整し直すことにしたのだが、その結果、図3-19のようなグラフを得た。 ちなみに、1440分は24時間（1日）を意味する。ゆで上げ後の待ち時間に関する調理実験では、最長24時間のスパゲッティを食したこともあり、それに由来して1440分とした。

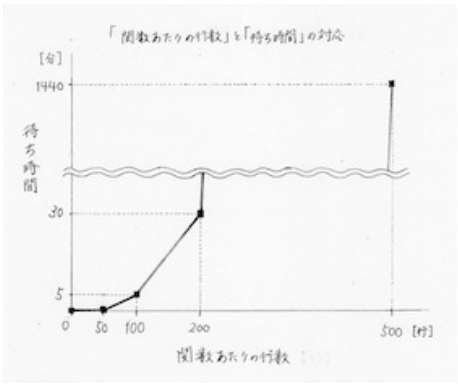


図3-19 「関数あたりの行数」に対する「作り置き時間」のグラフ

- 識別子の名称の長さ → スパゲッティ料理の名称（IdLength2SpaghettiName）

　これは、変数名の具体例を挙げながら対応付けを決めてゆく。

変数名が3字未満の場合、例えば「v」なら、何を表しているかなど一目では分からないため、ソースの種類も見当がつかない「スパA」という名称に対応付けた。

　次に3字以上5字未満の場合、例えば「val」なら、かろうじて「値（value）」と解釈できるかもしれないが、本当に「値」の意味だろうかという不安と、どのような「値」なのかという疑問を与えるものであろう。 本当に「スパゲッティ」なのかという不安や、「トマト」はソースなのか具なのかという疑問を与える「トマトスパ」に対応付けた。

　同様に5字以上10字未満の場合、例えば「value」や「maxValue」なら、それが「値」であることが確実であり、また補足的な意味も表現できる。 そこで、簡潔・確実にその意味を伝えられる「トマトソースのスパゲッティ」に対応付けた。

　10字以上20字未満の場合、例えば「maxValueOnList」なら、どのような値なのかという条件を的確に伝えられるが、やや冗長な印象もある。 セールスポイントを積極的に表現する「有機栽培トマトのこだわりスパゲッティ」に対応付けた。

　最後に20字以上の場合、例えば「maxValueOnNewPriceList」なら、細かなニュアンスまで把握できるが、目で全体を追わねば理解に至らない。 雰囲気の演出に拘るあまり読みづらい（注文しづらい）名称「賀茂茄子と香草のスパゲッティ アンデス風 サルサ・ディ・ポモドーロを添えて」に対応づける。

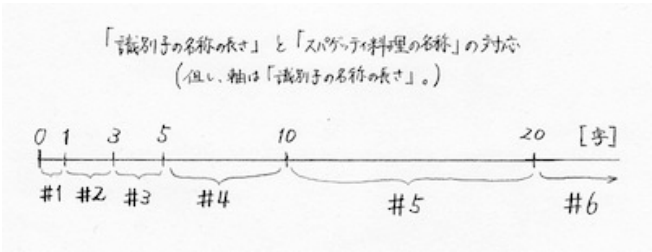


図3-20 「識別子の名称の長さ」に対する「スパゲッティ料理の名称」のグラフ

#	スパゲッティ料理の名称
#1	？？？
#2	スパA
#3	トマトスパ
#4	トマトソースのスパゲッティ
#5	有機栽培トマトのこだわりスパゲッティ
#6	賀茂茄子と香草のスパゲッティ　アンデス風　サルサ・ディ・ポモドーロを添えて

- インデントのずれ具合 → ゆで時間のばらつき（IndentDifference2VariabilityOfBoilingTime）

  一行あたりのインデントのずれ具合は最小値「0」が良い値である。そしてゆで時間のばらつき（の幅）も最小値「0」が良い値である。 つまり両者共に0を原点として比例の関係にあると考える。そこで、比例定数を用いて次のように算出することにした。

  ゆで時間のばらつき[分] ＝ 比例定数 × インデントのずれ具合[文字]

  関数あたりの行数や識別子の名称の長さなどは、常軌を逸しない限り、つまりある程度は自由に決めることができる。 しかし、インデントはそうでなく、およそ一意に定まるものであり、少しのばらつきも許容されるべきではないものと考える。 例えば、10行の中で1字分ずれていた場合（ずれ具合の値が0.1だった場合）は、やや気にかかる程度として、ゆで時間のばらつきを1分とする。 つまり、比例定数を「10」に設定するのである。

  アル・デンテを作り出そうとすると、ゆで時間1分のずれは死活問題となる。 同じように良いプログラムを書こうとすると、10行に1字のずれも許されないということになる。 少し厳しい気もするが、インデントに対する意識が欠落している学生が多いことから、その戒めとしてこの値を設定する。

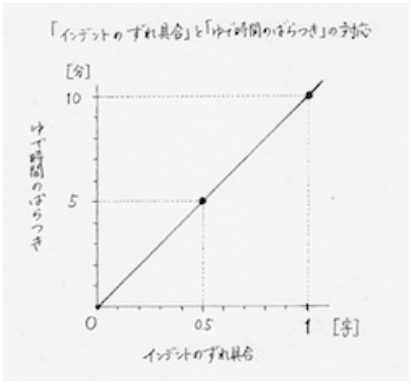


図3-21 「インデントのずれ具合」に対する「ゆで時間のばらつき」のグラフ

- コメントの記述量 → オリーブオイル量（CommentRatio2OliveOil）

  10%未満の割り合いであれば、無視できる程度のコメントであるとして、オリーブオイルは「なし」とする。 10%以上20%未満なら、たまに目に入れる程度（香り付け程度）のコメントであるとして「少量」とする。 20%以上50%未満なら、少なくもなく多くもなくで「適量」とする。 50%以上80%未満なら、プログラムがコメントに埋もれて、その読解に支障を来たしかねないため、適量を超えて「大量」とする。 80%以上付されていようものなら、もはやそれはプログラムを読むのではなく、コメントを読んでいるようなものである。つまりオリーブオイル（コメント）が主体になりかねない状態であり「オイル漬け」とする。

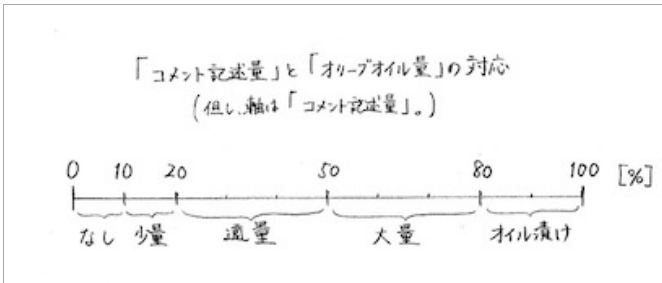


図3-22 「コメントの記述量」に対する「オリーブオイル量」のグラフ

なお、第7期開発は、2013年1月7日から同月8日まで実施した。

第8期：予備実験後の再調整

第4章 実験でも述べるが、予備実験実施後の検討で、再調整を要すると判断したメトリクスプラグインがある。以下に説明する。

- 関数あたりの行数 → 作り置き時間（LinesPerFunction2WaitTime）

  作り置き時間による食感の変化が当初の予定よりも小さく、良し悪しが分かりづらいという結果が得られた。 これにより、さらに作り置き時間を長く設定することにして、先のグラフを図3-23のように変更した。 本実験では、このグラフの通りの写像を行うことにする。

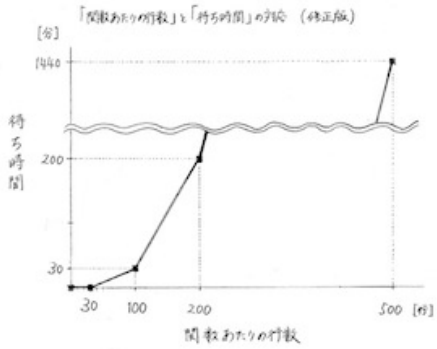


図3-23 「関数あたりの行数」に対する「作り置き時間」のグラフ（訂正版）

なお、第8期開発は、予備実験と同日である2013年1月15日に実施した。

現状での実績

成果

現段階における成果について述べる。 当初の計画で定めた期間を超過してしまったが、写像役として提起した「メトリクスプラグイン」と、メトリクスグラフを表現するための「SynapseScript」を、それぞれ開発することができた。 これらを用いたメトリクスの実例をご覧に入れたい。

- 割り算の実例

  身近な写像例として除算（割り算）を挙げ、具体的に「100 ÷ 3」の商を算出して示そう。

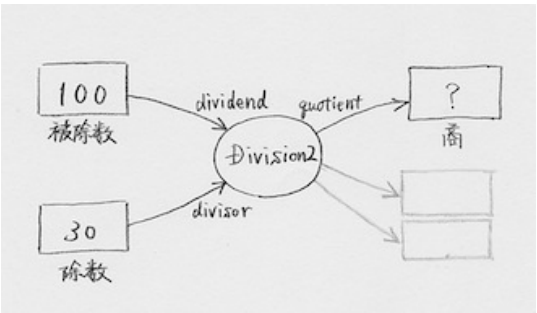


図3-24 割り算を表すメトリクスグラフ

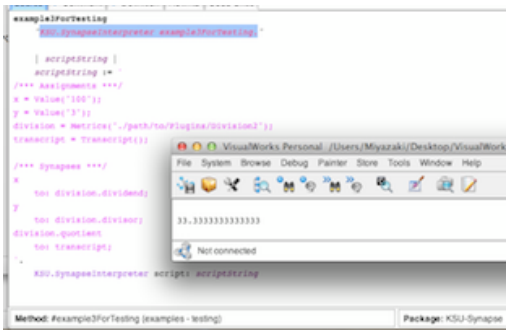


図3-25 割り算の結果（商）を得たところ

```
/*** Assignments ***/
x = Value("100");
y = Value("3");
division = Metrics("./path/to/Plugins/Division2");
transcript = Transcript();

/*** Synapses ***/
x
    to: division.dividend;
y
    to: division.divisor;
division.quotient
    to: transcript;
```

商が得られるのは、トランスクリプト（図3-25の中央に位置するウィンドウ）への出力に「division.quotient」を充てているからだ。 その部分を「division.floorQuotient」とすれば整数化された商が得られ、また「division.remainder」とすれば剰余を得ることもできる。

ちなみに「Division2」という名称は、Base64エンコードを必要としなかった頃のプラグイン「Division」に対するものである。

- 可食化の実例

  本研究の第一義である、プログラムからスパゲッティへの「可食化」を実現する様子を示そう。

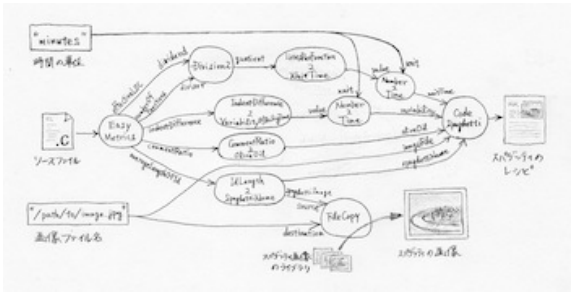


図3-26 可食化を実現するメトリクスグラフ

図3-27 スパゲッティのレシピ

```
/*** Assignments ***/
sourceFile         = File("./path/to/Programs/source.c", "read");
htmlFile           = File("/fullpath/to/CodeSpaghetti.html", "write");
imageFilename      = Value("/fullpath/to/CodeSpaghetti.jpg");
fileCopy           = Metrics("./path/to/Plugins/FileCopy");
easyMetrics        = Metrics("./path/to/Plugins/C_EasyMetrics");
division           = Metrics("./path/to/Plugins/Division2");
lpf2WaitTime       = Metrics("./path/to/Plugins/LinesPerFunction2WaitTime");
id2Spaghetti       = Metrics("./path/to/Plugins/IdLength2SpaghettiName");
indent2Variability = Metrics("./path/to/Plugins/IndentDifference2VariabilityOfBoilingTime");
comment2OliveOil   = Metrics("./path/to/Plugins/CommentRatio2OliveOil");
codeSpaghetti      = Metrics("./path/to/Plugins/CodeSpaghetti");
timeUnit           = Value("minutes");
number2Time_1      = Metrics("./path/to/Plugins/Number2Time");
number2Time_2      = Metrics("./path/to/Plugins/Number2Time");

/*** Synapses ***/
sourceFile
    to: easyMetrics;
easyMetrics
    send: effectiveLOC to: division.dividend
    send: quotient     to: lpf2WaitTime
    to: number2Time_1.value
    to: codeSpaghetti.waitTime;
easyMetrics
    send: numberOfFunctions to: division.divisor;
easyMetrics
    send: averageLengthOfId to: id2Spaghetti
    send: spaghettiName     to: codeSpaghetti.spaghettiName;
easyMetrics
    send: indentDifference to: indent2Variability
    to: number2Time_2.value
    to: codeSpaghetti.variability;
easyMetrics
    send: commentRatio to: comment2OliveOil
    to: codeSpaghetti.oliveOil;
id2Spaghetti
    send: spaghettiImage to: fileCopy.source;
imageFilename
    to: fileCopy.destination;
imageFilename
    to: codeSpaghetti.imageFile;
timeUnit
    to: number2Time_1.unit;
timeUnit
    to: number2Time_2.unit;
codeSpaghetti
    to: htmlFile;
```

第6期開発で定めたグラフ（図3-15）とは異なることに注意されたい。 当時のグラフ構造を実装する予定だったが、画像データをBase64エンコードに掛けて受け渡しを行うと、コマンド文字列が長すぎるという理由（シェルの制約）により実行できなかったのだ。 そこで、データそのものを受け渡すのではなく、そのデータの在り処（つまり画像ファイルへのパス）を用いて間接的に参照し、プラグイン「FileCopy」によるコピー操作で実現することができた。

同じく、図3-15に存在しなかったプラグイン「Number2Time」を新たに用いているが、これは「待ち時間」や「ゆで時間のばらつき」の出力値を読みやすい形式に変換するためのものである。 これら「FileCopy」や「Number2Time」の詳細については、それぞれのヘルプ機能（make help）をご覧頂きたい。

- 従来型を模したメトリクスツールの実例

  従来のメトリクスツール（得られた指標値を一覧表示するもの）に似せた例を示そう。なお、やや煩雑になることからメトリクスグラフは割愛する。

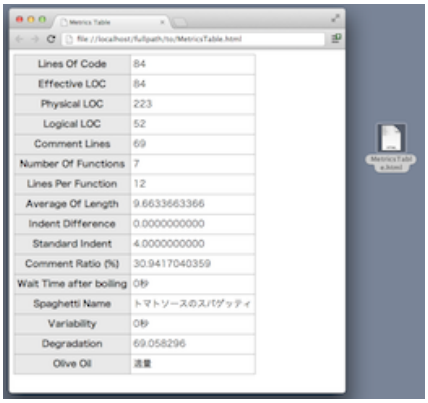


図3-28 従来型に似た結果提示（一覧表示）

```
/*** Assignments ***/
sourceFile          = File("./path/to/Programs/source.c", "read");
htmlFile            = File("/fullpath/to/MetricsTable.html", "write");
easyMetrics         = Metrics("./path/to/Plugins/C_EasyMetrics");
division            = Metrics("./path/to/Plugins/Division2");
lpf2WaitTime        = Metrics("./path/to/Plugins/LinesPerFunction2WaitTime");
id2Spaghetti        = Metrics("./path/to/Plugins/IdLength2SpaghettiName");
indent2Variability  = Metrics("./path/to/Plugins/IndentDifference2VariabilityOfBoilingTime");
comment2Degradation = Metrics("./path/to/Plugins/CommentRatio2Degradation");
comment2OliveOil    = Metrics("./path/to/Plugins/CommentRatio2OliveOil");
metricsTable        = Metrics("./path/to/Plugins/MetricsTable");
timeUnit            = Value("minutes");
number2Time_1       = Metrics("./path/to/Plugins/Number2Time");
number2Time_2       = Metrics("./path/to/Plugins/Number2Time");

/*** Synapses ***/
sourceFile
    to: easyMetrics;
easyMetrics.linesOfCode       to: metricsTable.linesOfCode;
easyMetrics.effectiveLOC      to: metricsTable.effectiveLOC;
easyMetrics.physicalLOC       to: metricsTable.physicalLOC;
easyMetrics.logicalLOC        to: metricsTable.logicalLOC;
easyMetrics.commentLines      to: metricsTable.commentLines;
easyMetrics.numberOfFunctions to: metricsTable.numberOfFunctions;
easyMetrics.averageLengthOfId to: metricsTable.averageLengthOfId;
easyMetrics.indentDifference  to: metricsTable.indentDifference;
easyMetrics.standardIndent    to: metricsTable.standardIndent;
easyMetrics.commentRatio      to: metricsTable.commentRatio;
easyMetrics
    send: effectiveLOC to: division.dividend
    send: quotient     to: lpf2WaitTime
    to: number2Time_1.value
    to: metricsTable.waitTime;
easyMetrics
    send: numberOfFunctions to: division.divisor;
division
    send: quotient to: metricsTable.linesPerFunction;
easyMetrics
    send: averageLengthOfId to: id2Spaghetti
    send: spaghettiName     to: metricsTable.spaghettiName;
easyMetrics
    send: indentDifference to: indent2Variability
    to: number2Time_2.value
    to: metricsTable.variability;
easyMetrics
    send: commentRatio to: comment2Degradation
    to: metricsTable.degradation;
easyMetrics
    send: commentRatio to: comment2OliveOil
```

```
    to: metricsTable.oliveOil;
timeUnit
    to: number2Time_1.unit;
timeUnit
    to: number2Time_2.unit;
metricsTable
    to: htmlFile;
```

- 三点見積もりの実例

　最後に、少しだけ実用的な例を示したい。仕事の現場で何か見積もりを行う際、楽観的に見た時の数値・悲観的に見た時の数値・最頻値（標準値）の三つから、見積もりを行おうというものである。 ここでは、その見積もり期待値の算出を例示しよう。

```
見積もり期待値 ＝（楽観値 ＋ 最頻値×４ ＋ 悲観値）÷ ６
```

種別	ラベル名	説明
入力	optimistic	楽観的に見た時の数値。楽観値。
入力	standard	悲観的に見た時の数値。悲観値。
入力	pessimism	最頻値。標準値。
出力	expectation	見積もりの期待値。
出力	variance	見積もりの分散。
出力	probability	見積もりの精度

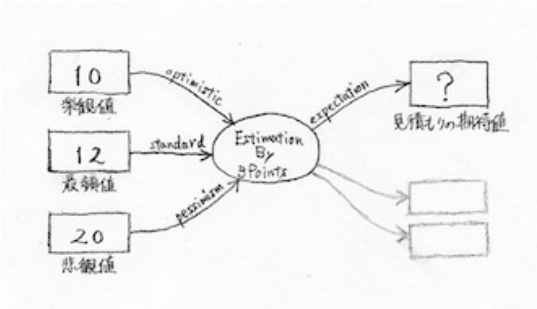


図3-29 三点見積もりのメトリクスグラフ

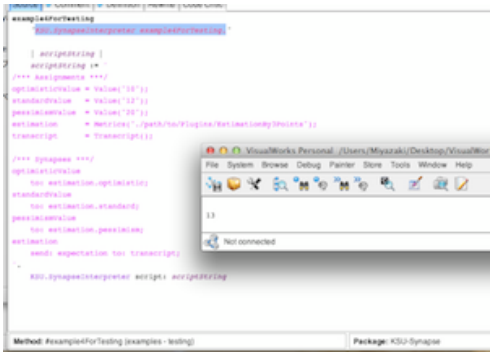


図3-30 得られた見積もり期待値

```
/*** Assignments ***/
optimisticValue = Value("10");
standardValue   = Value("12");
pessimismValue  = Value("20");
estimation      = Metrics("./path/to/Plugins/EstimationBy3Points");
transcript      = Transcript();

/*** Synapses ***/
optimisticValue
    to: estimation.optimistic;
standardValue
    to: estimation.standard;
pessimismValue
    to: estimation.pessimism;
estimation
    send: expectation to: transcript;
```

　以上、四種類のメトリクスが実現されたことを示した。「プログラムからスパゲティへの可食化の実例」によって可食化が実現できたことを示し、 その他の数例を示すことによって再利用性の高いツールが実現できたことを示せたであろう。

　なお、このメトリクスツール（及び上記四題のメトリクス）を利用するには、以下の構成要素が不可欠となる。参考として、開発時に利用していたバージョンや条件を括弧内に付しておく。

> - VisualWorks (7.8 with じゅん for Smalltalk)
> - make (GNU Make 3.81)
> - gcc (4.2.1)
> - awk (20070501)

- javac, java (1.7.0_09)
- Perl (5.12.4)
- PHP (5.3.15)
- Python (2.7.2)
- ruby (1.8.7)
- sh (GNU bash 3.2.48)

問題と今後

現段階のメトリクスツールが持つ問題点と今後の対処について述べる。

- 分散処理に対応できていない問題

  メトリクスグラフの評価（写像処理の実行）は、写像元からのデータ供給に始まり、メトリクスプラグインによる写像を経て、写像先まで到達する。 このうちの先頭である「写像元からのデータ供給」について、スパゲッティへの可食化の例ように複数の写像元が存在し得るのだが、その場合、そのどちらも並行にデータ供給を開始することができるはずである。 しかし、開発したSynapseScriptの処理系は、その評価を全て「逐次処理」で実装しており、より効率のよい「分散処理」で実現することが望まれる。 ただし、その実現には、分散システムの学域で議論される諸問題（プロセス間の干渉など）への対処を新たに追加する必要がある。

- 循環構造（再帰構造）を表現する場合の問題

  もし、メトリクスグラフで循環構造（再帰構造）を表現することができれば、フィボナッチ数の算出やニュートン法など、再帰的な構造を持つ計算処理を実現することもできよう。

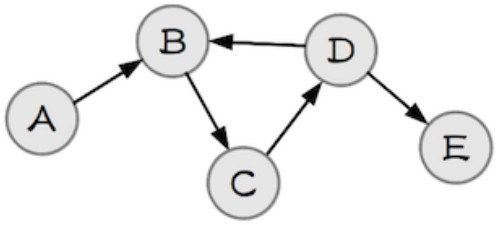


図3-31 循環構造を含むメトリクスグラフ

しかし、問題点が存在する。再帰構造に入ったところ（図3-31のノードB）で、必要な入力値二つを得られないため、その先の評価が恒久に行なわれないのである。 プログラミングにおける再帰構造では、再帰構造に入った後に、その中で記述されている再帰構造に遷移するのだが、 メトリクスグラフの場合は、再帰構造に入る前に、中の再帰構造の評価結果を要するのである。つまり、再帰を表現することにメトリクスグラフは向いていないものと考えられる。 この問題点をツール側の工夫で解決することはできないものか、今後の検討で打開案を見出すことが期待される。

動作確認

開発したメトリクスツールの検証を目的に、以下に挙げる環境で動作確認を実施した。

- MacBook 13" [Late 2008] (Apple)

  2013年2月2日、下記マシンにてメトリクスツールの正常動作を確認した。また、このマシンを利用してツールの開発を行なったということも記しておく。

  - OS: OS X 10.8.2
  - Processor: Intel Core 2 Duo 2GHz
  - Memory: 1GB→2GB→4GB
  - NVIDIA GeForce 9400M (256MB)
  - Internal Solid State Drive: 256GB (SSD)

- iMac 27" [Late 2009] (Apple)

  2013年2月2日、下記マシンにてメトリクスツールの正常動作を確認した。

	• OS: OS X 10.8.2 • Processor: Intel Core i5 2.66GHz (QUAD-CORE) • Memory: 4GB→8GB • ATI Radeon HD 4850（512MB) • Internal Hard Disk Drive: 1TB Serial ATA @ 7200rpm

- MacBook Pro 15" [Late 2008] (Apple)

  2013年2月2日、下記マシンにてメトリクスツールの正常動作を確認した。

	• OS: Mac OS X Lion 10.7.5 • Processor: Intel Core 2 Duo 2.93GHz • Memory: 4GB • NVIDIA GeForce 9600M GT (512MB) • Internal Solid State Drive: 256GB (SSD)

学生証番号：947424　氏名：宮崎 雅文

第4章 実験

本章では、本研究の一環として実施する「スパゲッティの試食評価実験」について述べる。

概要

目的

本研究の第一義である「可食化」の提案にあたり、その有効性を検証するため当実験を執り行う。 従来型の「可視化」と比較した時の「可食化」の優位性を立証し、論旨主張の根拠にすることを目的とする。

構成

上記の目的を果たすため、可視化と可食化の比較を行う。 従来用いられてきた手法である「可視化」と比較して優位な評価を得ることができたなら、その「可食化」の有効性も示すことができよう、と考えたのである。 ただし、その比較や評価は人間の感覚に基づくものであり、つまり人間による評価を要する。そこで、当実験では被験者にご協力いただき、可視化及び可食化に関する評価実験を行うことにした。

まず、可食化の実例として、ソフトウェアプログラムをスパゲッティとして表現することを示す。 そのスパゲッティを試食してもらい、被験者自身が感じ取った「スパゲッティの良し悪し」を評価してもらう。 次に、可視化の実例として、プログラムのメトリクス結果を従来の方法（一覧表示）で提示する。 その計測結果を通して、被験者自身が感じ取った「プログラムの良し悪し」を評価してもらう。 双方の詳しい手順は次節で述べるとして、この二部構成の後、被験者には「可視化」と「可食化」のそれぞれに対する評価を求める。 その評価結果を統計的に分析・検定し、「直感的な理解・認知」における可食化の優位性を示すことが、当実験の目標である。

当実験は、実施に関する問題点を洗い出すための予備実験と、それら問題点を改めて実施する本実験とで構成する。 なお、「当実験」とは、本章で述べる一連の実験を指し、また「本実験」とは、予備実験に対する本番の実験を指すものとする。

対象の限定

序論では、可食化に対して、次のようなねらいを定めていた。

> *可食化が有効な情報表現手段であることを実証し、その新たな活用の可能性を見出すこと。*

可食化が「情報表現手段」として有効であることを示したい、と述べたのだが、やや抽象的な表現であり、その対象は広域に渡る。 広く一般に

「有効である」と述べることができるのならば、この表現でも良いのだが、その実証もまた広域を網羅する形で行なわなければならず、いささか現実的ではない。 そのため、より限定した対象への有効性を述べようと考え、その対象を「直感的な理解・認知」に絞り込んだのだ。 つまり、当実験では、直感的な理解・認知における「可食化」の有効性を検証することになる。

手順

「可食化」の例題提示とその評価

開発したメトリクスツールを用いて、ソフトウェアプログラムをスパゲッティとして可食化し、被験者にはその試食を通してアンケート記入をお願いする。 被験者に試食してもらうスパゲッティには四種類あり、それぞれを個別に評価してもらうのだが、その違いは、先の章で述べた「スパゲッティの指標」の値の違いである。

#（対応するプログラム）	オリーブオイル量	待ち時間	ゆで時間ばらつき
一皿目（01_Normal）	適量【良い】	0分【良い】	0分【良い】
二皿目（02_BadComment）	なし【悪い】	0分【良い】	0分【良い】
三皿目（03_BadLPF）	適量【良い】	1分47秒【悪い】	0分【良い】
四皿目（04_BadIndent）	適量【良い】	0分【良い】	6分18分【悪い】

- 一皿目のスパゲッティ（良いスパゲッティ）

  最初に試食してもらうスパゲッティは、全ての指標が良い結果を示す場合のもの（良いスパゲッティ）である。ただし、これが「良いスパゲッティ」であるということは被験者に伝えない。 事前知識（先入観）がない状態で試食してもらい、その上で「直感的にどのような印象を持ったか」という観点で、良し悪し七段階からの択一で評価してもらう。 ちなみに、この後の三種類についても、同様の観点で評価してもらうものとする。また、七段階評価とは別に自由記述欄を設け、その評価に至った理由を書いてもらう。

  事前に評価基準を示すこともなく試食してもらうため、被験者の評価には「ばらつき」が生じるはずである。 この「ばらつき」を抑えるためには、やはり評価基準を示す必要があり、それは可視化と同様であると結論づけたい。 つまり、可視化と可食化の前提条件を揃えることで、後の議論を用意にすることがねらいである。

  なお、この一皿目のスパゲッティを基準として、以後の実験項目を評価してもらいたい。その旨は試食後の被験者に伝達するものとする。

- 二皿目のスパゲッティ（オリーブオイルのない悪いスパゲッティ）

  二皿目として、オリーブオイルを絡めていない「乾燥しやすいスパゲッティ」を試食・評価してもらう。 一皿目と同様に、良し悪しを七段階で評価してもらうが、それ以外に「どの指標が最も特徴的だったか」という質問を用意しており、 アンケート用紙に記載する四つの語群（指標の名称）の中から一つ選び出すことになる。 その上で、選択した理由を自由記述欄に書いてもらう。この新しい評価項目は、この後の二種類についても同様に存在する。

  もちろん、期待することは「悪い」という評価である。また、スパゲッティの表面光沢や乾燥によるまとまりを理由として指標「オリーブオイル量」を選択するはずである。 この予想が的中するならば、スパゲッティへの可食化の指標として「オリーブオイル量」を用いることが適切だったと言えよう。

- 三皿目のスパゲッティ（待ち時間付きの悪いスパゲッティ）

  三皿目として、ゆで上げから時間が経過して乾燥してしまった、俗にいう「作り置きのスパゲッティ」を試食・評価してもらう。 二皿目と同様に、七段階評価及び特徴的な指標の選択を求める。

  これも「悪い」という評価が与えられると期待する。また、乾燥によるまとまりを理由として指標「ゆで上げ後の待ち時間」が選択されるだろう。 この予想が的中するならば、二皿目と同様に「ゆで上げ後の待ち時間」という指標が適切だったと言えよう。

- 四皿目のスパゲッティ（ゆで時間がばらついている悪いスパゲッティ）

  四皿目として、ゆで時間の短いものと長いものとが混在する「食感のばらつきがあるスパゲッティ」を試食・評価してもらう。 これは、二皿目や三皿目と同様の評価を求める。

  同じく「悪い」という評価が与えられよう。また、柔らかいものと芯の残るものが混ざっていることを理由に指標「ゆで具合のばらつき」が選択されるはずだ。 この予想が的中するならば、「ゆで具合のばらつき」が適切な指標だったと言えよう。

- 五皿目（スパゲッティ料理の名称の評価）

  五皿目は、実際のスパゲッティとして提供するわけではない。スパゲッティの指標として提案した「スパゲッティ料理名」の有効性を評価してもらうための実験項目である。 ただ、これは単なる名称であり、直接的にスパゲッティの食感に影響を与えるものではないため、可食化の指標とすることに疑問を感じるかもしれないが、 少なくとも、これまで数々のスパゲッティ料理店を渡り歩いた経験によると、料理名は、食感に対するバイアスを与えるものとして充分な役割を果たすものである。 従って、「スパゲッティ料理名」を評価に値するものとし、当実験の評価項目に含めることとした。

  次に示す三種類のスパゲッティ料理名を提示し、その名称に感じる「短い」または「長い」といった印象を、七段階で評価してもらう。

#	スパゲッティ料理名
一つ目	スパA
二つ目	トマトソースのスパゲッティ
三つ目	賀茂茄子と香草のスパゲッティ アンデス風 サルサ・ディ・ポモドーロを添えて

一つ目は「短い」、二つ目はどちらでもない、三つ目は「長い」といった評価が期待される。 この予想と一致するならば、食感に影響を与えうる指標（可食化の指標）としての有効性が認められよう。

以上で、可食化に関する実験の手順が完了する。なお、この可食化の写像元となる「対応するプログラム」は、付録資料[5]として添えてある。

「可視化」の例題提示とその評価

比較対象として挙げた「可視化」に関する例題として、ソフトウェアプログラムのメトリクス結果を従来型の一覧表示により提示し、プログラムの良し悪しを評価してもらう。 提示するメトリクス結果を二種類用意し、それぞれに対して「良いプログラム」「どちらとも言えない」「悪いプログラム」の三者択一の質問を行い、 その評価に至った理由を自由記述欄に記載してもらう。

図4-1 一覧表示型のメトリクス結果（一つ目）

図4-2 一覧表示型のメトリクス結果（二つ目）

- 一つ目（インデントのばらつきがある悪いプログラム）

  被験者には、一覧表示型のメトリクス結果（図4-1）を見てもらい、入力プログラムの良し悪しを評価してもらう。 この例では、インデントのばらつきが「0.629...」となっており、インデントがばらついている「悪いプログラム」であることが分かる。

  ただ、その指標の意味を理解していなければ、「0.629...」という数値の良し悪しは判定が付かないのではないか。従って、正しく判定できる被験者は少ないと予想する。

- 二つ目（良いプログラム）

  二つ目の例（図4-2）は、いずれの指標値にも大した問題点がない「良いプログラム」のメトリクス結果である。 ただ、一つ目と同じように、直感的に「良い」と感じることができるだろうか。スパゲッティのように直感的に「美味しい（良い）」と感じることはできないであろう。 従って、「良いプログラム」であることを認識できる被験者は少ないものと予想する。

以上で、可視化に関する実験の手順が完了する。

総括的な評価

上記の例題を終えた後、「直感的な理解・認知」という観点から「可食化」及び「可視化」を評価してもらう。 具体的には、それぞれの手法に対して十点満点の評価を求め、また、その点数で評価した理由を自由記述欄に記入してもらう。 ただ、可食化の場合は「スパゲッティの良し悪し」を評価してもらうのであり、その写像元である「プログラムの良し悪し」については触れていない。 そこで、両者への評価を前に、図4-3を被験者に示すことで、可食化の評価対象を「スパゲッティ」から「プログラム」に変更してもらう。 つまり、可食化及び可視化の評価対象を、共に「プログラム」に揃えた上で評価してもらうことにする。

図4-3 プログラムとスパゲッティの対応関係

実施

予備実験

2013年1月15日、被験者1名にご協力いただいて予備実験を執り行った。 その中で気付いたことや被験者から頂戴したご指摘のうち、重要な事項のみ以下に説明する。

- 調理のむらが生じる

  実際に調理してみると、慣れていないこともあって調理のむら（芯の残り具合が一定でないなど）が生じてしまう。 この問題を解消するため、実施者である私自身も試食し、その時々の調理むらを把握して、実験後に調整することにした。

- 「ゆで上げ後の待ち時間」が分かりづらい

  対応するプログラム（03_BadLPF）を可食化した際、ゆで上げ後の待ち時間として「1分47秒」という指標値が得られたため、その結果に従って予備実験を実施した。 ところが、時間が短かったのか、評価結果を見ても明確な効果が認められなかった。 そのため、関数あたりの行数から待ち時間を算出するプラグイン（LinesPerFunction2WaitTime）の写像具合を再度調整し、より長い待ち時間が与えられるようにした。 修正の後には「10分43秒」という指標値を得たが、これは以前実施した「待ち時間に関する調理実験」の結果を鑑みても、スパゲッティが絡みまとまる待ち時間であることは明らかだった。 従って、本実験では後者の結果「10分43秒」を採用することにする。

- オリーブオイルによる干渉が生じる

  「オリーブオイル量」の加減が変化すると、別の指標である「ゆで上げ後の待ち時間」への干渉が認められた。 本来、待ち時間が長い場合、スパゲッティが乾燥してまとまるのだが、オリーブオイルを絡めていると、ほとんどまとまらない。 干渉の程度も甚だしいため、「ゆで上げ後の待ち時間」に関する実験項目においてオリーブオイルは絡めないことにする。 （この場合の指標「オリーブオイル量」の取り扱いについては、後の考察で別途説明する。）

  ちなみに「オリーブオイルを絡めない」とした場合のプログラムは、付録資料[5]の中の「03_BadLPF2」に相当する。 コメントを取り除くことで、オリーブオイル量を「なし」にしたのである。

#（対応するプログラム）	オリーブオイル量	待ち時間	ゆで時間ばらつき
三皿目（03_BadLPF2）	なし【悪い】	10分43秒【悪い】	0分【良い】

この予備実験での失敗・気付きを活かし、有意義な本実験になるよう配慮する。

本実験

2013年1月16日、被験者7名（うち1名途中退席）にご協力いただいて本実験を執り行った。 問題が生じないように配慮していたが、被験者からご指摘を頂戴した。

- 基準となるスパゲッティの評価について

  一皿目は「良いスパゲッティ」として提示したのだが、これを基準にしてしまうと、残りのスパゲッティは必然的に「悪いスパゲッティ」になってしまう。 当初は「どちらとも言えない」を原点として良し悪しを評価してもらおうと考えていたのだが、指摘を受けて初めて、その原点が「良いスパゲッティ」になっていると気付かされたのだ。 この点を勘違い（また曖昧に）していたことは、当実験における反省点である。

  実験時の説明通り、一皿目を「良いスパゲッティ」として（つまり「良いスパゲッティ」を原点として）評価してもらうことについて、特に問題は生じないはずである。 しかし、その質問の表現を工夫する余地はあっただろう。「良し悪し」を問うのではなく「悪さの程度」を問うべきであった。実験前に気付かなかったことが悔やまれる。

結果

「可食化」の実験項目

ここでは、可食化の各実験項目で得られた被験者による評価結果について述べる。 合計で四種類のスパゲッティを試食してもらったが、それぞれに対して良し悪しの七段階（とても良い・良い・少し良い・どちらとも言えない・少し悪い・悪い・とても悪い）からの択一で評価を求めた。このデータは順序尺度であるから、その代表値として中央値ならびに四分位偏差を採用する。その際、各指標値に対して以下のように便宜上の数値を割り当てる。

1	2	3	4	5	6	7
とても良い	良い	少し良い	どちらとも言えない	少し悪い	悪い	とても悪い

各実験項目について、それぞれ中央値と四分位偏差などを求め、またグラフ（箱ひげ図）も提示する。

- 【Q1】一皿目（良いスパゲッティ）の評価

図4-4 【Q1】の箱ひげ図

- 中央値：2
- 四分位偏差：1
- 四分位範囲：2
- 最小：2
- 最大：5

- 【Q4-被験者全員】二皿目（オリーブオイルのない悪いスパゲッティ）の評価

図4-5 【Q4-被験者全員】の箱ひげ図

- 中央値：2
- 四分位偏差：2.5
- 四分位範囲：5
- 最小：1
- 最大：6

- 【Q4-正答者】二皿目（オリーブオイルのない悪いスパゲッティ）の評価

正答率（最も特徴的な指標として「オリーブオイル量」を選択した被験者の割り合い）は、28.6%であった。

図4-6 【Q4-正答者】の箱ひげ図

- 中央値：4
- 四分位偏差：2
- 四分位範囲：4
- 最小：2
- 最大：6

- 【Q6-被験者全員】三皿目（待ち時間付きの悪いスパゲッティ）の評価

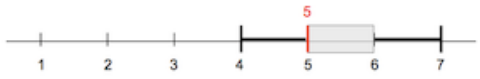


図4-7 【Q6-被験者全員】の箱ひげ図

- 中央値：5
- 四分位偏差：0.5
- 四分位範囲：1
- 最小：4
- 最大：7

- 【Q6-正答者】三皿目（待ち時間付きの悪いスパゲッティ）の評価

正答率（最も特徴的な指標として「ゆで上げ後の待ち時間」を選択した被験者の割り合い）は、83.3%であった。

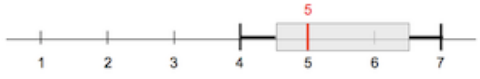


図4-8 【Q6-正答者】の箱ひげ図

- 中央値：5
- 四分位偏差：1
- 四分位範囲：2
- 最小：4
- 最大：7

- 【Q8-被験者全員】四皿目（ゆで時間がばらついている悪いスパゲッティ）の評価

- 中央値：6

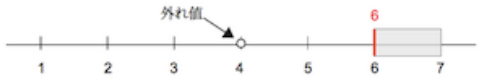


図4-9 【Q8-被験者全員】の箱ひげ図

- 四分位偏差：0.5
- 四分位範囲：1
- 最小：4
- 最大：7

- 【Q8-正答者】四皿目（ゆで時間がばらついている悪いスパゲッティ）の評価

  正答率（最も特徴的な指標として「ゆで具合のばらつき」を選択した被験者の割り合い）は、83.3%であった。

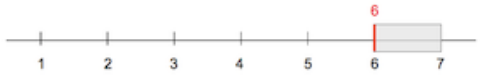


図4-10 【Q8-正答者】の箱ひげ図

- 中央値：6
- 四分位偏差：0.5
- 四分位範囲：1
- 最小：6
- 最大：7

また、四種類の試食を終えた後に「スパゲッティ料理名」に対する評価を求めた。これも同様に七段階（とても短い・短い・やや短い・どちらとも言えない・やや長い・長い・とても長い）からの択一で評価してもらった。 このデータも順序尺度であり、中央値と四分位偏差を採用する。以下は、便宜上の数値の割り当てである。

1	2	3	4	5	6	7
とても短い	短い	やや短い	どちらとも言えない	やや長い	長い	とても長い

先程と同様に、中央値や四分位偏差などを算出し、対応するグラフ（箱ひげ図）を示す。

- 【Q9】料理名「スパA」への評価

図4-11 【Q9】の箱ひげ図

- 中央値：1
- 四分位偏差：0
- 四分位範囲：0
- 最小：1
- 最大：2

- 【Q10】料理名「トマトソースのスパゲッティ」への評価

図4-12 【Q10】の箱ひげ図

- 中央値：4
- 四分位偏差：0
- 四分位範囲：0
- 最小：3
- 最大：4

- 【Q11】料理名「賀茂茄子と香草のスパゲッティ アンデス風 サルサ・ディ・ポモドーロを添えて」への評価

図4-13 【Q9】の箱ひげ図

- 中央値：7
- 四分位偏差：0
- 四分位範囲：0
- 最小：6
- 最大：7

「可視化」の実験項目

次に、可視化の各実験項目で得られた被験者による評価結果について述べる。 従来の一覧表示で提示（可視化）するメトリクス結果を見て、プログラムの良し悪しを三段階（良いプログラム・どちらとも言えない・悪いプログラム）の択一で評価してもらう。 このデータは順序尺度であり、代表値として中央値ならびに四分位偏差を採用する。 各指標値に対する便宜上の数値は、以下のように割り当てた。

1	2	3
良いプログラム	どちらとも言えない	悪いプログラム

提示するメトリクス結果は二種類ある。それぞれについて中央値・四分位偏差・グラフ（箱ひげ図）を提示する。

- 【Q12】一つ目（インデントのばらつきがある悪いプログラム）への評価

図4-14 【Q12】の箱ひげ図

- 中央値：2
- 四分位偏差：0.5
- 四分位範囲：1
- 最小：1
- 最大：2

- 【Q13】二つ目（良いプログラム）への評価

図4-15 【Q13】の箱ひげ図

- 中央値：1.5
- 四分位偏差：0.5
- 四分位範囲：1
- 最小：1
- 最大：2

「可視化」と「可食化」の評価

可視化及び可食化の実例をひと通り提示し終えたところで、最後にそれぞれの手法に対する十点満点の評価を求める。 実際のアンケート用紙には、次のような設問文を記載している。

> ソフトウェアプログラムの問題点や改善点を発見する上で、メトリクス結果の可視化および可食化はそれぞれ、どれくらい「直感的な理解・認知」を得られる手法でしょうか。

- 【Q14】「可視化」への評価

上記の設問に対する「可視化」への評価（十点満点）は以下の通りである。図4-16の左側を参照されたい。

- 平均値：3.83
- 標準偏差：2.79

- 【Q14】「可食化」への評価

同様に、「可食化」への評価（十点満点）は以下の通りである。図4-16の右側を参照されたい。

- 平均値：6.67
- 標準偏差：1.75

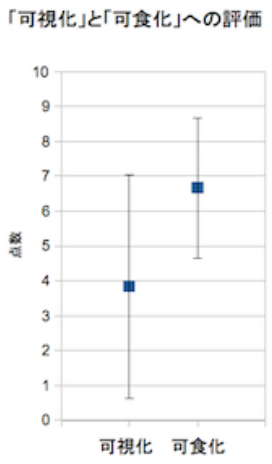


図4-16 「可視化」と「可食化」への評価

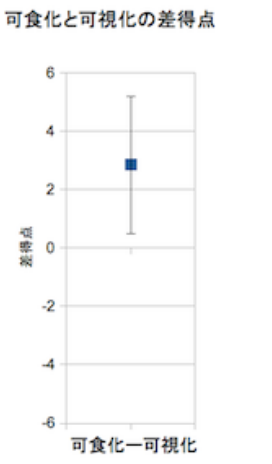


図4-17 可食化-可視化の差得点

- 【Q14】「可視化」と「可食化」の比較 —— t検定による

「可視化」と「可食化」の双方に評価が与えられたので、その評価結果を比較したい。 具体的には、それぞれの平均値同士に有意差があるか否かを明らかにしたい。 そこで、統計的仮説検定の一「t検定」を援用して、群間の差得点を検定することにより、有意差の有無を検証する。

> 対立仮説（H1）：両者の平均点に差がある。
> 帰無仮説（H0）：両者の平均点に差がない。（平均点の差が0に等しい）

この検定における仮説を上記の通り設定し、また、判断水準（有意水準）を両側5%に設定する。 この場合の自由度dfは、データ数（つまり被験者数6名）から次のように求められる。

```
自由度 df = (データ数 - 1) = (6 - 1) = 5
```

自由度5におけるt分布の臨界値（有意水準 両側5%）は「2.57」である。（cf. 参考文献[13]） また、検定統計量tの算出に必要な変数値を以下に示す。「可食化への評価」から「可視化への評価」を引いた「差得点」を被験者ごとに算出した、差得点データに関する変数値である。 図4-17も参照されたい。

- 標本平均 M : 2.83
- 母平均 μ : 0（平均点に差がないという仮説を立てているため）
- 標準偏差 s : 2.04
- 被験者数 n : 6

ここで、検定統計量tは、次のように求めることができる。ここでsqrt(x)とは、xの平方根を表す。

```
検定統計量 t = ((M - μ) / (s / sqrt(n - 1))) = 3.11
```

検定統計量t「3.11」が、臨界値「2.57」の外側にあることから、帰無仮説（H0）は棄却される。 従って、対立仮説（H1）が採択され、5%水準で平均点同士に有意差があると言える。

考察

以上の結果に基づいて考察を行う。

「可食化」の実験項目

- 【Q1】予想よりばらつきが少ない

  一皿目（良いスパゲッティ）について、事前知識・評価基準が明確になっていないことから、評価にばらつきが見られると予想していたのだが、 評価結果【Q1】を見ると、さほどのばらつきが見られない。 確かに、実際に提供したスパゲッティは特段の欠点が存在していたわけでもないため、無難な評価を選ぶ被験者が多かったのだと考えられる。 自由記述欄にも「突飛したものがなく評価できない」といった言葉が見受けられた。 また、「好み」という言葉で表現されていたり、「お腹がすいていたのでおいしく感じた」など、被験者の嗜好や体調なども大きく影響を与えることがわかる。

- 【Q4】指標「オリーブオイル量」は評価が曖昧になる

  評価結果【Q4-被験者全員】や【Q4-正答者】を見ると、評価のばらつきが大きいとわかる。 オリーブオイル量が「なし」になり、乾燥しやすく、まとまりやすいスパゲッティであるから、当初の予想では、悪い評価が与えられると考えていた。 しかし、そもそも「オリーブオイル量」に着目した被験者の割り合い（正答率）自体が低かったことや、正答していても評価のばらつきが大きいことから、 （もちろん、他の指標との組み合わせもあるが、）スパゲッティへの可食化において指標「オリーブオイル量」は不適切だったとも考えられる。

  また、自由記述欄には「乾燥していてはしで持ち上げると周りも全部ついてくる」という趣旨の意見が見られ、予想通りの着眼を得たのだが、その被験者が選択した指標は「ゆで上げ後の待ち時間」であった。 オリーブオイルを使わない場合は乾燥しやすいスパゲッティになるが、ゆで上げ後の待ち時間が長い場合は乾燥したスパゲッティになる。 意味合いは異なるのだが、結果として現れる状態は、どちらも「乾燥したスパゲッティ」なのだ。指標同士が干渉を起こし、正しい判断ができなかったものと考えられる。

- 【Q6】【Q8】指標「ゆで上げ後の待ち時間」「ゆで具合のばらつき」の結果は予想に近い

  【Q6-被験者全員】や【Q6-正答者】、【Q8-被験者全員】や【Q8-正答者】の評価結果を見ると、 そのどちらも、ほとんどの被験者が悪い評価を与えていることがわかる。特に「ゆで具合のばらつき」の結果は顕著であろう。 従って、スパゲッティへの可食化において指標「ゆで上げ後の待ち時間」「ゆで具合のばらつき」は適切であったと考えられる。

- 【Q9】【Q10】【Q11】「スパゲッティ料理名」に対する印象は予想通り

  当初の予想とほぼ一致し、ばらつきもなかったことから、指標「スパゲッティ料理名」によって印象（先入観）を付与できることがわかる。ただ、この指標については試食評価を行なわず、名称への評価のみを実施したため、可食化の指標として有効であることを示すには、根拠が不足しているだろう。 今後の課題としたい。

  その上で、自由記述欄の内容から抜粋して補足したい。高級感に関する言及が散見されることから、食感への影響も考えられるのではないだろうか。

  - 【Q9】に対する自由記述欄
    - 弁当箱に入っているスパゲッティより酷い。
    - 安いイメージ。高級感などは感じられない。

- 【Q10】に対する自由記述欄
  - 馴染み深く、無難な長さ。
  - 自宅で作るパスタ。
- 【Q11】に対する自由記述欄
  - スパゲッティ専門店のよう。高そう。
  - 注文前に、ある程度どんなものか想像することができる。
  - 長いので言いづらそう。高級感は感じられる。

「可視化」の実験項目

- 【Q12】【Q13】可視化でも正しく判定できている

  【Q12】と【Q13】について、ばらつきは同じであったが、中央値を見ると、確かにプログラムの良し悪しと相関していることがわかる。従って、可視化によってプログラムの良し悪しを判定できることが言える。 また、自由記述欄には、「メトリクス結果を見る気がしない。良し悪しが分からない」「数字だけからは何とも言えない」などといった意見が見られたが、 同時に「比較するものがないので。項目の意味が分からず、上の条件に合てはまるかわからない」という意見もあった。 事前に指標の意味を説明していないため、その数値を見たところで良し悪しが判定できないということである。 つまり、対象分野の知識を持つことを前提にしなければ、可視化ではその良し悪しを判定できないのだと考えられる。

  ただ、どうして三段階評価にしてしまったのか、実施しておきながら後悔している。 可視化した情報から判別できる段階数は多くないだろう、と勝手に判断したことが間違っていた。これまでと同じように七段階で評価してもらうべきだっただろう。

「可視化」と「可食化」の評価

- 【Q14】直感的な理解・認知において、有意差をつけて可食化が有利

  評価結果【Q14】を見るに、「可視化」「可食化」両者の平均点には差があるように見える。 ただし、評価のばらつきによっては有意な差と言えないかもしれないため、統計的な仮説検定により実証することにした。 その結果、確かに有意差があると認められ、直感的な理解・認知において「可視化」よりも「可食化」が有効であることが示された。 つまり、当初の論旨を主張するための根拠が得られたと言える。

  この設問に対する自由記述欄の内容については、総括で触れることにする。

総括

ほとんどの被験者が同様の意見を呈していたのだが、その長短を含めて分かりやすくまとめられた意見があるので、そのまま引用したい。

> *可視化の方はパッと見ても標準が分からないので全然良いのか悪いのか分かりませんでした。“初学者”という対象を考えると、数で見る結果より、直接感じられる可食化が良いと思います。可食化は、人によって感じ方が違ったり、難しいところも多いと思いますが、直感的な理解にはとても効果的だと思います。*

可視化（数値化）は、基準を与えなければ良し悪しを判断することができない。言い換えれば、基準を与えさえすれば（その指標の意味を理解していなくとも）その良し悪しを正確に判断できるのだ。 しかし、それは可食化とて同じことである。基準となるスパゲッティを知らなければ、結局のところ良し悪しを判断することはできない。 また、嗜好や体調によっても感じ方は異なり、安定した情報提示が必要となる場面では使い物にならないであろう。誤認識を生み出しかねない。

しかし、例えば不案内な分野（上の「初学者」で言えば「プログラミングの分野」）を理解しなければならない場合、最初から厳密・正確な把握が必要だろうか。まずはその概観を把握すべく、大雑把な理解に留めることもあろう。 その際に、いくら厳密で正確な指標値を見せられたとしても、その指標の意味を理解していなければ良し悪しも判断できず、何も意味を成さない。 そこで可食化を持ち出せばどうだろう。指標の意味や数値（指標値）の良し悪しを理解できなくとも、より身近な「食べ物」という存在を介して、大雑把に理解すれば良いではないか。

もちろん、今回の実験で指標同士の干渉が問題になったように、どのような状況でも一概に有効であるとは決して言えないが、それら制約を解消して利用するならば、可食化によって直感的な理解・認知を得られたこともまた事実である。 従って、当初の論旨と比較するとやや限定的ではあるが、直感的な理解・認知において「可食化」は有効な情報表現手段であると言えよう。

学生証番号：947424　氏名：宮崎 雅文

第5章 結論

本章では、序論で提起した各論旨に対する結論を述べる。

「可食化」の提案

要約

序論にも掲げたように、「可食化」の提案に対するねらいは次の通りである。

> *可食化が有効な情報表現手段であることを実証し、その新たな活用の可能性を見出すこと。*

このねらいを実現するため、本研究では被験者にご協力を仰いで評価実験を行い、その有効性の実証を試みた。 従来型の情報提示手段である「可視化」と新しく提案する「可食化」の双方について「直感的な理解・認識」という視点で評価してもらい、その結果として「可食化」の優位性が認められた。 これにより、当初の論旨に比べてやや限定的ではあるが、その有効性を実証することができたのである。

結論

従って、本論旨に対して以下のように結論づけることができる。

> 直感的な理解・認識を必要とする場合、可食化という手段は有効である。

反証可能性

この主張は、少なくとも次に挙げる事項を前提条件とするものであり、万一その前提が崩壊した場合、上記の結論を再現できない可能性がある。

- 「可視化」の有効性
  実験で「可食化」の有効性を実証する際、その比較対象として「可視化」を挙げた。 「可視化に対する評価」と比較した際の「可食化に対する評価」の優位性を根拠にして実証を謳っている。 つまり、「可視化」が情報提示に対して有効な手段であることを前提としているため、「可視化」の有効性が否定されるなら、今回の実証は意味を成さない。
- 写像（対応付け）の妥当性
  プログラムの指標値をスパゲッティの指標値として写像したが、その対応付けは全て、これまでの調査や調理実験の結果から得た経験則によるものである。 それも、研究者（私）一人の体験に基づくものであり、極めて限定的な経験則であると言わざるをえない。 つまり、この経験則に致命的な欠陥が存在するならば、今回の実証は意味を成さない。

今後の課題

本研究の第一義を果たすことができたのだが、問題点が残されていることも否めない。以下に、今後の課題として示しておく。

- 写像（対応付け）への根拠の獲得
  前項にも記したが、プログラムとスパゲッティとの対応付けが妥当であるとするには、とても根拠に乏しい。 そのため、今後さらに実験を重ねることで、確かな経験則を得なければならない。 ただ、対応付けが正しいとして可食化の妥当性を調べるのか、逆に可食化が有効である

として対応付けの妥当性を調べるのか、という問題もある。 何かしらの打開策はないものだろうか。

メトリクスの新しい枠組みの提案

要約

序論では、メトリクスを実現するための新しい枠組みの提案について、次のようなねらいを定めていた。

> *メトリクスツールのための新しい枠組みを提案し、メトリクスツールの改善によってユーザの活用を促進させること。*

このねらいを実現するため、提案した枠組みを援用するメトリクスツールを実際に開発し、可食化やその他のメトリクスの実例を提示した。 枠組みを用いた実例を示すことにより、その適応性の改善を実証したものとする。 また、ツールを利用する状況に応じて必要な指標値のみを提示できたことから、その冗長性も改善できたものと考える。

結論

従って、本論旨に対して以下のように結論づけることができる。

> 新しい枠組み「メトリクスグラフ」により、従来のメトリクスツールの冗長性及び適応性を改善することができた。

反証可能性

前節に同じく、以下に挙げる前提条件が崩壊した場合は、上記の結論を再現できない可能性がある。

- メトリクスグラフの万能性

  致命的な問題にはならないことだが、念の為に触れておく。 従来のメトリクスツールを抽象化し「メトリクスグラフ」という構造を見出し、このグラフによってあらゆるメトリクスを表現することができると論じたが、 その可能性を何かしらの理論に基づいて証明したわけではない。つまり、本論旨には客観的な根拠が与えられていないのである。 この不都合を取り払うべく、可食化に限らず他の具体的なメトリクス例も示し、「あらゆるメトリクス」を実現することの論拠に替えたかったのだが、それら例題も網羅的とは言えない。 よって、万一「メトリクスグラフ」によって表現できないメトリクスが存在したとすれば、本論旨に含まれる「適応性の改善」を一概に主張することはできない。

今後の課題

本論旨にまつわる問題点について、以下の通り今後の課題として設定する。

- メトリクスグラフの定式化

  本研究で、新しい枠組み「メトリクスグラフ」を提案したが、その論拠には主観的かつ曖昧な部分が多い。 特に、前項でも述べた通り「あらゆるメトリクス」を本当に実現できるのか、その点を証明し、概念として定式化すべきである。

- メトリクスグラフの普及

  本研究では、新しい枠組みを提案したに過ぎず、具体的な活用事例は上記の「可食化」以外に存在しない。 これに留めず枠組みを広く普及させ、ねらいで述べた「ユーザによるメトリクスツール活用の促進」を実現しなければならない。 そのために、開発したメトリクスツールを「枠組みの活用事例」とし、世界に向けて公開したい。

- 開発したメトリクスツールの改善

  可食化の実現と枠組みの活用を目的として開発したメトリクスツールだが、第3章 道具と作成物でも言及した通り、その理論や実現方法に多くの問題を抱えている。 時に「生き物」と喩えられるほど、その変貌が目覚ましい「ソフトウェア」である。落ち着きを覚えず絶え間なく、その時々の要求に応じ改良を重ねてゆくことが望まれる。

学生証番号：947424　氏名：宮崎 雅文

おわりに

あとがき

「はじめに」で示した三つの興味をやや強引に絡めつつも、コンピュータ科学史で用いられてきた「スパゲッティプログラム」という汚名を返上すべく研究を進めてきたが、 それ果たすことができただろうか。こればかりは、提案する可食化や枠組みの普及を待つほかないが、ひとまず、自身の考えを主張できたことは評価に値するだろうと考えている。

思い返せば、私はいつも受動的で、何事も誰かに指示されなければ行動に移せなかったものだが、この研究を通して、そんな自分自身の新たな可能性を見出せたように思う。 まさか、スパゲッティを卒業研究の題材にするなど思ってもみなかったのだから。 時間が掛かってしまうかもしれない、思い至らないこともあるかもしれない、それでも「できるかもしれない」という気持ちを常に心の隅に置いていたために、この成果物を収めることができたのだと思う。

こうして満足のゆく学生生活を送ることができ、とても幸せに感じている。

さて、完全な蛇足になるが、論旨の一「メトリクスツールの枠組みの提案」に関連して、一つ述べておきたいことがある。やや差し出がましい話になってしまうが、ご容赦を賜りたい。 従来のメトリクスツールはあらゆる情報を詰め込んでいる、と先の章で説明したが、そうなることも無理はない。 あらゆる立場のユーザに対応しようとすれば、その全てのユーザが求める機能を詰め込めこまねばならないのだから。 しかし、どうしてユーザ自身でもない開発者が、いつも頭を悩ませなければならないのか。そう疑問に思ったことがある。

> *餅は餅屋*

何事もその道の専門家に頼るべし、という意味のことわざだが、実は提案した枠組みはこの言葉をライトモチーフにして構想したものである。「どのような指標値を必要としているのか」は、開発者ではなくユーザ自身の知るところであって、その本人が考えるべきことではないのか。 そう考えて、ユーザ自身がメトリクスの構造を再現できるような仕組みづくりを行なったのだ。

「人にやさしいソフトウェア」とは何か。人を怠けさせるためのソフトウェアのことだろうか。否。人が持ち合わせる本来の能力を存分に活かせるように設計された「ソフトウェア」を指すべきではないか。 ユーザ自身が頭で考えて、その結果として便利な生活を実現する。こうした「考えることで得をする」という構造が必要だ。 間違っても「何もしなくとも得をする」といった幻想や、独りよがりの「やさしさ」など持ち出すべきではない。さもなくば、また私のように「受動的な人間」が生まれることになろう。

無論、ソフトウェアに限った話ではなく、あらゆる製品にも同じ事が言えるはずである。 上辺だけではなく、真の意味での「豊かな生活」を手に入れようとするなら、目を背けず一度考えてみていただきたい。

最後に、不遜な言葉を書き連ねたことについてお詫び申し上げ、あとがきの言葉に代えようと思う。

謝辞

本研究の遂行にあたり、実に多くの方々からご支援を賜った。 指導教官の青木 淳教授には、研究のことに限らず広範囲に渡ってご指南を頂戴し、目標を見失いかけていた人生に鋭気を取り戻すことができた。 先輩の水野 信さんには、たくさんの良い実例を見せていただき、研究へのモチベーションを維持することができた。 また、同輩の青木 優知さんには、いつも的確な助言を呈してもらい、研究活動というドライブを安全に進めることができた。 御三方には特に深謝を申し上げたい。

そして、長時間に渡る評価実験に付き合ってくださった同じく青木研究室の皆様、またお忙しい中アンケート調査にご協力くださった京都産業大学の皆様とSmalltalk勉強会@京都の皆様、 プログラミングの演習科目で拵えたプログラム一式を惜しみなく提供してくれた二人の友人に対して、感謝の意を示したい。 殊に、オムロン株式会社の濱崎 治さんには、実装に必要となるDLLCCという仕組みを懇切にご教示いただいた。記名により改めてその謝意を表したい。

研究を名目に身勝手な振る舞いを繰り返し、家族には常に迷惑を掛けてきた。その謝罪とともに、いつも見守っていてくれたことに感謝したい。ありがとう。

手に本を取ると、どうしても「はじめに」から一字一句飛ばさずに読もうとする癖がある。 おかげで本を最後まで読めた例など数えるほどしかないのだが、その分、その著者が記した謝辞を何度も読むものだから印象に残っている。 よくあれほどの人名を挙げて長々と書けるものだ、などと感じていたが、ここに来てようやく、その著者らの気持ちを理解することができた。 大袈裟かもしれないが改めて、私がここに生きることを支

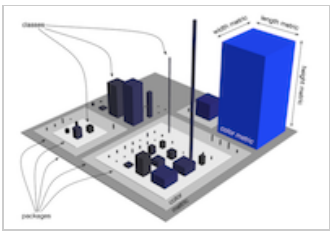
えてくださった全ての皆様に心から感謝の意を申し上げたい。

学生証番号：947424　氏名：宮崎 雅文

参考文献

関連研究

[1] Software Systems as Cities



- Richard Wettel, Michele Lanza, and Romain Robbes　著
- URL: http://www.inf.usi.ch/phd/wettel/publications.html
- 最終確認日: 2013年2月5日

パスタ関連

[3] パスタの歴史

- シルヴァーノ・セルヴェンティ、フランソワーズ・サバン 著
- 飯塚茂雄、小矢島聡　監修
- 清水由貴子　翻訳
- 原書房
- 2012年
- ISBN: 9784562047536

[4] パスタ百科 ― 新しい感覚で作る決定版（暮しの設計 NO.205）

- 中央公論新社
- 1992年
- ISBN: 9784128000777

[5] イタリア・パスタおいしい物語

- 鈴木 奈月　著
- 東京書籍
- 2002年
- ISBN: 9784487798636

メトリクス関連

[14] Smalltalkイディオム

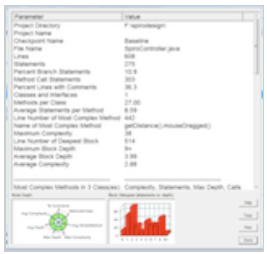
- 青木 淳　著
- ソフトリサーチセンター
- 1997年
- ISBN: 9784915778810

その他

[2] ソフトウェア作法

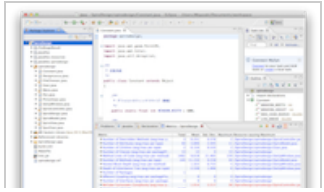
- Brian W.Kernighan、P.J.Plauger　著
- 木村 泉　著
- 共立出版
- 1981年
- ISBN: 9784320021426

[6] SourceMonitor



- URL: http://www.campwoodsw.com/sourcemonitor.html
- 最終確認日: 2013年2月5日

[7] Eclipse Metrics plugin



- URL: http://sourceforge.net/projects/metrics/
- 最終確認日: 2013年2月5日

[8] レポート・論文の書き方入門

- 河野 哲也　著
- 慶應義塾大学出版会
- 2002年
- ISBN: 9784766409697

[9] XP エクストリーム・プログラミング入門 —— ソフトウェア開発の究極の手法

- Kent Beck　著
- 長瀬 嘉秀、飯塚 麻理香、永田 渉　翻訳
- ピアソンエデュケーション
- 2000年
- ISBN: 9784894712751

[10] はじめてのコンパイラ 原理と実践

- 宮本 衛市　著
- 森北出版
- 2007年
- ISBN: 9784627817210

[11] コンパイラ 情報処理シリーズ(7)

- A.V.エイホ、J.D.ウルマン　著
- 土居 範久　著
- 培風館
- 1986年
- ISBN: 9784563007874

[12] VisualWorks DLL and C Connect User's Guide

- URL: http://www.cincomsmalltalk.com/pdf/DLLandCConnectGuide.pdf
- 最終確認日: 2013年2月5日

[13] これならわかる 確率・統計セミナー

- 森本 宏明、大橋 守　著
- 培風館
- 1997年
- ISBN: 9784873611761

学生証番号：947424　氏名：宮崎 雅文

付録資料

本研究に関係する付録資料の紹介を行う。

プログラムの性質を表す指標の一覧

- メトリクスツール「Resource Standard Metrics (RSM)」における指標

  以下に掲載する指標群は、メトリクスツール「Resource Standard Metrics (RSM)」のウェブサイトに掲載されていたものからの抜粋である。

  - http://msquaredtechnologies.com/m2rsm/docs/rsm_metrics.htm [最終確認日：2013年2月13日]
  - http://msquaredtechnologies.com/m2rsm/docs/rsm_metrics_narration.htm [最終確認日：2013年2月13日]

指標名	説明
LOC	Lines of Code。コードの行数。SLOC（Source LOC）とも。計測の規則も様々で、以下にはその例を幾つか掲載する。関数毎・クラス毎・パッケージ毎で計測する。
Effective LOC	eLOCとも。空行・コメント行・括弧だけの行を除いてカウントした数値。
Physical LOC	物理LOC。ソースファイルに記載されいているプログラムの行数をそのまま計測した数値。
Logical LOC	論理LOC。lLOCとも。プログラムにおける文（statement）の数を表す数値。1行に複数文書かれている場合も文の数だけカウントする。
Comments Lines	コメントの行数。
Blank Lines	空行の行数。
FP (LOC由来)	LOCの値から算出したFP（Function Point：ファンクションポイント）値。機能数とその複雑さによって算出した数値。
Number of Input Parameters	引数の数。
Number of Return Points	関数が戻る場所（return）の数。
Interface Complexity	Input Parameters + Return Points。インタフェースの複雑さ。
Cyclomatic Complexity	循環的複雑度。線形独立な経路の数。循環の数 + 1。
{Functional, Class, Namespace/Package} Complexity	Interface + Cyclomatic。
Average, Maximum, Minimum	関数ごとのLOCについて算出する平均値・最大値・最小値。
Number of Data attributes	属性（フィールド）の数。アクセス修飾子（public, private, protected）毎に数え分ける。
Number of Methods	操作（メソッド）の数。アクセス修飾子（public, private, protected）毎に数え分ける。
Number of Functions	関数（メソッド？）の数。クラス毎・パッケージ毎で計測する。
Depth of Inheritance Tree	継承の階層数。
Number of Classes	クラス数。
Number of Base Classes	基底クラス（親クラス）の数。
Number of Derived classes	派生クラス（子クラス）の数。
Derived/Base Class Ratio	派生クラス数と基底クラス数の比率。派生クラス数÷基底クラス数で求める。
{Maximun, Average} Inheritance Depth	継承の階層数の最大値・平均値。
{Maximun, Average} Number of Child Classes	子クラス数の最大値・平均値

Classes	
Average functions per class	クラス毎の関数（メソッド）数。
Comment and White space percentages	プログラム中のコメントや空白が占める割合。
Average Character line length	1行の文字数の平均値。
Memory Allocation and De-allocation metrics	メモリの動的確保及び解放を行なっている箇所の数。
Language Keyword use	利用している予約語（キーワード）の数。
Language Construct use	利用している言語構造の数。（PHPのprintやexitのようなもの。）
Extract Comments per file for understandability rating and spell checking	コメントの理解しやすさとスペルチェック。
Extract Strings per file for spell checking	文字列のスペルチェック。
Metrics differentials between two file version	2バージョンのファイルについてメトリクス結果の比較を行う。
Total C, C++ and Header Files	C, C++のソースファイル及びヘッダファイルの総数。
Total Java Files	Javaのソースファイルの総数。
Total Number of Files	総ファイル数。

- メトリクスツール「Source Monitor」における指標

  以下は、Javaプログラムを解析にかけた場合の指標群だが、指標名と実測値からその意味を類推しているため、誤訳の可能性を否定出来ない。

指標名	説明
Lines	空行を除いた行数を表す数値。
Statements	セミコロンで終わる文と、クラスやメソッドの定義文の合計を表す数値。
% Branches	おそらく、Statements中に存在するif文の割合を表す数値。
Calls	メソッド呼び出しの回数を表す数値。
% Comments	コメントが占める割合。
Classes	プログラム中のクラス定義数（内部クラス数を含む）を表す数値。
Methods/Class	メソッド数とクラス数の比率。メソッド数÷クラス数で算出する。
Avg Stmts/Method	1メソッドあたりの平均Statements数。
Max Complexity	おそらく、各メソッドのCyclomatic Complexityのうち、その最大のものを表す数値。
Max Depth	おそらく、{ }（ブロック）の最大ネスト数。
Avg Depth	おそらく、{ }（ブロック）の平均ネスト数。
Avg Complexity	おそらく、各メソッドのCyclomatic Complexityの平均値。

- メトリクスプラグイン「Eclipse Metrics plugin」における指標

指標名	説明
NORM	Number of Overridden Methods。オーバーライドしているメソッドの数。
NOF	Number of Attributes (Field)。属性（フィールド）数。
NSC	Number of Children。子クラス数。
NOC	Number of Classes。総クラス数。
MLOC	Method Lines of Code。メソッドのLOC。
NOM	Number of Methods。メソッド数。
NBD	Nested Block Depth。ブロックのネスト数。
DIT	Depth of Inheritance Tree。継承の階層数。
NOP	Number of Packages。パッケージ数。
CA	Afferent Coupling。依存しているパッケージ外のクラス数。
NOI	Number of Interfaces。インタフェースの数。
VG	MaCabe Cyclomatic Complexity。循環的複雑度。
TLOC	Total Lines of Code。コードの行数。
RMI	Instability。（パッケージの）不安定性。Ce / (Ca + Ce)で算出する。

PAR	Number of Parameters。引数の数。
LCOM	Lack of Cohesion of Methods。あるクラスの凝集性の欠如を表す。小さいほど凝集度は大きく、メソッドの強度が高いことを表す。
CE	Efferent Coupling。依存しているパッケージ内のクラス数。
NSM	Number of Static Methods。スタティックメソッド（クラスメソッド）の数。
RMD	Normalized Distance。
RMA	Abstractness。パッケージの抽象度。全クラス及びインタフェース中の抽象クラス及び抽象インタフェースの割合。
SIX	Specialization Index。クラスの特殊化指標の平均値。NORM * DIT / NOMで算出する。
WMC	Weighted methods per Class。あるクラスに定義されているメソッドのCyclomatic Complexityの総和。大きいほど複雑であり、メンテナンスのコストがかかることを示唆する。
NSF	Number of Static Attributes。スタティックフィールド（クラスフィールド）の数。

- その他の指標

指標名	説明
NOA	Number of Attribute。NOFに同じ。
NCV	Number of Class Variables。クラス変数の数を表す。NSFに同じ。
NIV	Number of Instance Variables。インスタンス変数の数を表す。
NCM	Number of Class Methods。クラスメソッドの数を表す。NSMに同じ。
NIM	Number of Instance Methods。インスタンスメソッドの数を表す。
NOC	Number of Children。サブクラスの数を表す。NSCに同じ。大きいほどサブクラスへの影響力が強く、メンテナンスに注意が必要であることを示唆する。
CBO	Coupling between Objects。あるクラスに関係しているクラスの数を表す。大きいほど他のクラスに依存していることを表し、複雑でメンテナンスコストがかかることを示唆する。
RFC	Response for a Class。あるクラスに関係しているメッセージの数を表す。大きいほど発信しなければならないメッセージが多いことを表し、複雑なクラスであることを示唆する。
HF	Hierarchy Factor。あるクラスの抽象性（具体性）を計測した値。0の場合は、そのクラスがインヘリタンス（クラス階層）の一番上に位置することを意味し、階層を下がるにしたがって1に近づく。0に近いほどクラスの抽象性が高く、再利用性が良好であることを示唆する。
RF	Reference Factor。あるクラスの全体性（部分性）を計測した値。クラスの相互参照関係（半順序）のどのあたりに位置するのかを表す。他のクラスを参照する割合が大きいほど1に近づき、0に近いほどクラスの独立性が高く部品として使われる可能性が高いことを示唆する。
PF	Polymorphism Factor。あるクラスの多相性（一義性）を計測した値。クラスが所有するメッセージが、他のクラスの所有するメッセージとどのくらい重なっているのかを表す。1に近づくほどメッセージが重なっていることを示し、ポリモーフィズムが有効に働いている可能性を示唆する。0に近づくほどメッセージが統一されていないことを示唆する。
CP	Class Popularity。あるクラスの人気度を計測した値。人気の高いクラスほど、他のクラスから利用されていることを表す。人気の高いクラスが集中しているほど、依存しているクラスが統一されていることを表し、良好なプログラムであることを示唆する。
MP	Message Popularity。あるメッセージの人気度を計測した値。CPと同様で、人気の高いメッセージが集中しているほど、発信されているメッセージが統一されていることを表し、良好なプログラムであることを示唆する。
走行時間	プログラムの走行時間。ベンチマークテストにも。

実際に食したスパゲッティの一覧

これまでに研究の一環として食してきたスパゲッティ（またはパスタ）を挙げておく。

#	名称
1	北山洋食カフェ 和蘭芹「生ハムとモッツァレラチーズのトマトソース スパゲッティ」
2	日清 カップ麺「スパ王 生タイプ ペペロンチーノ」
3	サークルKサンクス「ルベッタ 大皿 明太子スパゲッティ」
4	スーパーマーケット成城石井「無着色明太子の彩りパスタ」
5	自家製「ボルカノ スパゲティ 1.8mm / ナスのトマトーススパゲッティ」
6	Osteria SAKURA「小海老と九条ねぎのアーリオ・オーリオスパゲティ」
7	喫茶マウンテン「甘口抹茶小倉スパ」

8	日清フーズ 冷凍食品「マ・マー 大盛りスパゲティ ナポリタン」
9	自主調理「ディ・チェコ フィリッポ567 スパゲッテリ（約1.8mm） / バジルソーススパゲッティ」
10	自主調理「日清フーズ マ・マー スパゲティ 1.6mm / バジルソーススパゲッティ」
11	北山ランタン「一膳パスタランチ 日替りパスタ フレッシュトマトとズッキーニのパスタ」
12	ポポラマーマ「ほうれん草ベーコンジェノベーゼ」
13	自主調理「ディ・チェコ No.11 スパゲッティーニ（1.6mm） / ボンゴレソーススパゲッティ」
14	カプリチョーザ「なすとホウレン草のミートソーススパゲティ」
15	カプリチョーザ「ベーコンとクリームのスパゲティ カルボナーラ」
16	セカンドハウス「セカンドトマトソース」
17	自主調理「日清フーズ マ・マー スパゲティ 1.8mm / カルボナーラ と ミートソース（2種類）」
18	セカンドハウス「ポテトたらこ」
19	自主調理「ガロファロ No.15 フェットチーネ（???mm） / カルボナーラスパゲッティ」
20	サイゼリヤ「イカの墨入りスパゲッティ」
21	自主調理「バリラ No.5 スパゲッティ（1.7mm） / バジルソース と ミートソース（2種類）」
22	自主調理「ディ・チェコ No.9 カッペリーニ（0.9mm） / 納豆」
23	自主調理「ジュゼッペ コッコ No.34 リングイーネ / ツナしょうゆ風味」
24	自主調理「青の洞窟 スパゲッティ 1.7mm / ジェノベーゼバジルソース」
25	自主調理「地中海のパスタ SAINT MICHEL 1.7mm Spaghetti」（チュニジア産）
26	自主調理「オーマイ スパゲッティ 1.7mm」
27	自主調理「AGNESI（アネージ） SPAGHETTI n.3 1.7mm」
28	Jolly-Pasta（ジョリーパスタ）「スモークサーモンといくら」
29	自主調理「ジュゼッペ コッコ No.33 スパゲッティ」
30	自主調理「PECK SPAGHETTI 1.8mm」
31	日清食品 冷凍食品「スパ王 プレミアム 海老とトマトの入ったジェノベーゼ」
32	サンヨー食品 カップ麺「カルボナーラ フェットチーネ」
33	TOMATO&ONION「和風シソ明太子スパゲティ」
34	あるでん亭「なすトマト」
35	洋麺屋五右衛門「豚しゃぶとたっぷり野菜の胡麻ダレ仕立て」
36	NIFTY「ツナときのこのクリームパスタ」
37	自主調理「Besler スパゲッティ 1.7mm」（トルコ産）
38	自主調理「ディ・チェコ No.41 ペンネリガーテ」
39	自主調理「CASTIGLIONI（カスティリオーニ） スパゲッティ No.52 1.67mm」
40	アンチョビ「ベーコンとホウレン草のトマトソース」

メトリクスプラグイン開発ガイドライン

- メトリクスプラグイン開発ガイドライン

SynapseScript 言語マニュアル

- SynapseScript 言語マニュアル

例題プログラム 五種

- 設問ページ
- 01_Normal
- 02_BadComment
- 03_BadLPF
- 03_BadLPF2
- 04_BadIndent
- 05_BadIdentifier

開発したメトリクスツールの一式

- マニュアル（MetricsTool.zipの取り扱い）
- MetricsTool.zip
- ドックレット（プログラム一覧）
  - KSU-Shell
    - KSU.ShellInterface
    - KSU_ShellInterpreter
    - KSU_ShellProcessManager
  - KSU-Synapse
    - SynapseInterpreter
    - SynapseNeuralConnector
    - SynapseNeuron
    - SynapseParseAction
    - SynapseParseBlockTable
    - SynapseParseNode
    - SynapseParser
    - SynapseParseTable
    - SynapseParseTree
    - SynapseScanner
    - SynapseScannerState
    - SynapseScannerTable
    - SynapseToken
    - SynapseTokenStream

スパゲッティの試食評価実験のデータ

- スパゲッティの試食評価実験 アンケート用紙
- 実験時に用いたスライド
- 結果データ（予備実験1名分＋本実験6-7名分）

学生証番号：947424　氏名：宮崎 雅文

メトリクスプラグイン開発ガイドライン

はじめに

本稿は、メトリクスツール開発に向けた新しい枠組みの構成要素「メトリクスプラグイン」を開発するためのガイドラインである。 設計思想や仕様の紹介を通して、メトリクスプラグインの開発を促進することが目的である。

商用・非商用を含め、世界には様々なメトリクスツールが存在する。 しかし、その多くは指標値の一覧表示、もしくはグラフとして図示する程度のものであり、必ずしもユーザに対して有意義な情報を提供できるとは限らない。 （あまり直感的でない。） ただ、それら現存のツールに冗長さが見られるのも無理はない。 ツールを利用するユーザの立場ごとに、どの指標が必要か不必要かを判断しなければならないが、それはツールの開発者の知るところではない。 その結果、あらゆる指標値をそのまま全て列挙せざるを得ないのだろう。

この冗長さを打開すべく、メトリクス向けの新たな枠組みを用いた実装物「SynapseScript」「メトリクスプラグイン」が開発された。 入力値に対応した出力値を応答する「メトリクスプラグイン」を、ユーザが自分の用途に合わせて接続することで、ユーザ自身の必要とする情報を効率的に提供するのである。 ここで特筆すべきは、その「メトリクスプラグイン」をユーザ自身で新たに作成できる点である。 ユーザの思うようなプラグインが用意されていない場合、ユーザは自身の持てる技術を存分に活かしながら独自のプラグインを開発できるのだ。 そんな「自分でプラグインを開発しよう」とする開発者を助けるため、本ガイドラインを作成した。ご活用いただけるなら幸いである。

定義

ここでは、本ガイドラインに記載されている用語の定義を行う。

- SynapseScript
  関数の役割を果たす「メトリクスプラグイン」を接続するための仕組み。データ記述言語。
- 指標
  プログラムなどの評価対象を計測する側面。いわば「ものさし」に相当する。実例として「行数」や「複雑度」など。
- 入力（input）/ 出力（output）
  変換系（メトリクスプラグイン）がデータＡをデータＢに変換する時、変換前のデータ（つまりＡ）を入力と呼び、変換後のデータ（つまりＢ）を出力と呼ぶ。 また、この変換のため変換系に対してデータＡの提供を行うが、この行為自体も入力と呼び、変換系がデータＢを作り出す行為も出力と呼ぶ。
- ビルドツールmake
  プログラムからソフトウェアを構築（ビルド）するための複雑な手続きを、makeコマンドの利用により単純化するもの。
- プログラム
  ソフトウェアプログラム。コンピュータを操作するための言語表現。ソースコードとも。

- メトリクス

  プログラムなどの評価対象を、何らかの尺度・指標（ものさし）で計測することを言う。

- メトリクスプラグイン

  具体的なメトリクスの処理（写像処理）を行う機能単位。何らかの入力値に対応する出力値を応答する変換系（翻訳者）である。 例えば、プログラムを入力して、そのプログラムの行数を計測して出力するなどを実現するもの。

ガイドライン

技術的概要

メトリクスプラグインは、ある入力値を一定の規則に従って変換し、出力値を応答するものである。 より具体的には、ソフトウェアのビルドに用いられるツール「make」を転用して、プラグインとしてのデータ変換処理を実現する。 makeのマクロ定義によって情報を入力し、その変換処理の結果を標準出力に対してJSON形式で出力する。 また、makeコマンドで動作するため、SynapseScriptを経由せず単体で用いることも可能である。 （図中の橙色がmakeに相当する。）

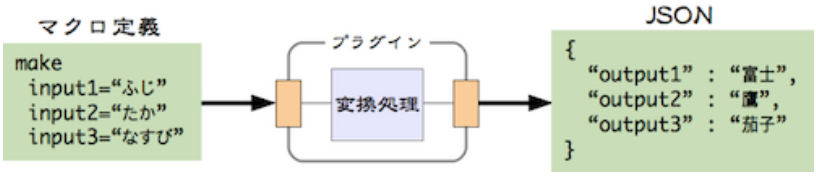


図1 プラグインのデータ入力とデータ出力

ただし、実際はこう単純には使えない。入力データや出力データに特殊文字が含まれている場合、そのエスケープ処理が必要になるのだが、 動作環境に依存しないように配慮しようとすると、そのエスケープ処理自体が煩雑になってしまう。そこで、回避策としてBase64エンコードを用いることにする。 プラグインの入出力に際してデータをそのまま扱うのではなく、Base64でエンコードした形で扱おうというものである。

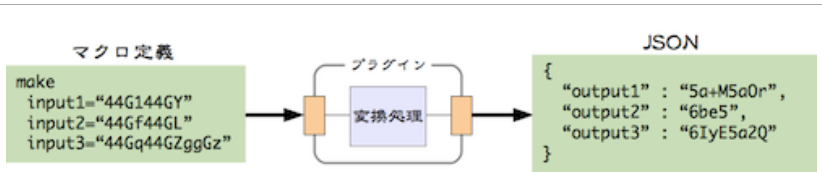


図2 プラグインのデータ入力とデータ出力（Base64利用）

エンコードやデコードの手間が掛かってしまうが、これで大方のデータを取り扱うことができる。ただし、makeが動作しているシェルの制約で、入力データの長さには限りがある。 巨大なデータを扱う場合には、データそのものを直接扱うのではなく、そのデータの在り処を示す情報を用いて間接参照するなど、工夫が必要になろう。 以上が、メトリクスプラグインの概要である。

ファイル構成

一つのメトリクスプラグインは、一つのディレクトリに対応する。以下には「Division2」というプラグインのファイル構成を示そう。

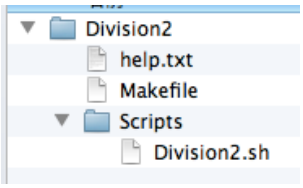


図3 プラグイン「Division2」のファイル構成

プラグインディレクトリ「Division2」の中に幾つかのファイルが含まれているが、この中の必須項目は「Makefile」のみであり、それ以外のファイルは全て任意項目である。 従って、メトリクスプラグインはその開発者によって自由に組み立てることができるのだ。

例題プラグイン（割り算）

図4 割り算のメトリクスグラフ

ファイル構成を示したメトリクスプラグイン「Division2」だが、端末（ターミナル）を介して実際に動かしてみよう。 必要であれば、こちら（Division2.zip）を解凍してご利用いただきたい。 プラグインディレクトリ「Division2」を用いて、割り算「100÷3」を実演するが、それに先立ち「100」と「3」をBase64エンコードに掛けておかねばならない。

```
$ printf "100" | base64
MTAw

$ printf "3" | base64
Mw==
```

「100」は「MTAw」、「3」は「Mw==」と表される。では、dividend（被除数）を「MTAw」、divisor（除数）を「Mw==」として、メトリクスプラグインによる割り算を実行してみよう。

```
$ cd /path/to/Division2
$ make dividend="MTAw" divisor="Mw=="
{"quotient" : "MzMuMzMzMzMzMzMzMzMzMw==", "floorQuotient" : "MzM=", "remainder" : "MQ=="}
```

quotient（商）として「MzMuMzMzMzMzMzMzMzMzMw==」、floorQuotient（整数化した商）として「MzM=」、remainder（剰余）として「MQ==」が得られた。 これをBase64でデコードしてみよう。

```
$ printf "MzMuMzMzMzMzMzMzMzMzMw==" | base64 --decode
33.3333333333333

$ printf "MzM=" | base64 --decode
33

$ printf "MQ==" | base64 --decode
1
```

それぞれ「33.3333333333333」「33」「1」と正しく計算されている。つまり、メトリクスプラグインによって割り算を行うことができた。以降では、この割り算プラグイン（Division2）を題材として説明する。

データの入力

メトリクスプラグインへのデータ入力は、次のようにして記述する。ここでは「xxxx」という4文字を入力する。

```
make input="xxxx"
```

これは最も単純な1入力の例であるが、先の割り算のように多入力の場合は次のように記述する。

```
make dividend="MTAw" divisor="Mw=="
```

上記2例の中で「input」「dividend」「divisor」と記している部分があるが、これはキー（makeの用語では「マクロ」）と呼ぶ。 等号「=」を挟んで、「"xxxx"」「"MTAw"」「"Mw=="」として引用符で囲んだ文字列を、値と呼ぶ。 値は、プラグインが利用される時に初めて具体的なデータを得るものであり、プラグインの開発時には何が渡されるか不定であるが、 キーはプラグインの開発者が独自に定めることができ、何が与えられるかわからない値を取り扱うための識別子となる。 キーと値のペアでデータ入力を行い、複数の入力が必要ならばスペースで区切って連ねることになる。

こうして渡された入力データは、Makefile内で「$(input)」「$(dividend)」「$(divisor)」と記述すれば利用できる。 あとは、この入力データを好きなように変換し、次節で述べる形式に従って出力するだけである。

データの出力

前節の例では「xxxx」という文字列を入力していたが、今度はその文字列をそのままの形で出力しよう。JSONという形式に従って以下のように出力される。

```
{ "output" : "xxxx" }
```

入力の際と同様に、ここでもキー「"output"」と値「"xxxx"」のペアで表現される。 ちなみに、入力の際はキーに引用符「"」が付かなかったが、出力の際は引用符を付ける。これはJSONの仕様によるものである。 次に示すものは、複数値を出力する例（割り算）である。「quotient」「floorQuotient」「remainder」の3ペアを

```
{"quotient" : "MzMuMzMzMzMzMzMzMzMzMw==", "floorQuotient" : "MzM=", "remainder" : "MQ=="}
```

入力のキーと同じく、出力のキーも開発者が独自に定めることができる。また、複数の値を出力するには、カンマ「,」で区切れば良い。

Makefile

割り算プラグイン「Division2」のMakefileの内容を以下に示そう。（一部修正）

```
#
# Division2 - Metrics for SynapseScript
#
#  Created by M.Miyazaki [2013.01.06]
#

all:
    @sh ./Scripts/Division2.sh "$(dividend)" "$(divisor)"

inputs:
    @echo dividend divisor

outputs:
    @echo quotient floorQuotient remainder

help:
    @less help.txt
```

簡単な記述であることがお分かりいただけるだろう。このMakefileには4つのターゲット「all」「inputs」「outputs」「help」が定義されている。それぞれを以下に説明する。

- all（デフォルトターゲット）

  入力値を出力値に変換する具体的な処理は、このターゲット「all」に記述する。これは必須ターゲットである。 この例では入力データ2つを、また別のシェルスクリプト（./Scripts/Division.sh）に渡している。割り算の計算は、そのシェルスクリプト内で記述されているのだ。 より詳細に述べれば、そのスクリプト内ではプログラミング言語Perlを用いて割り算の処理を行なっている。 もちろん、計算処理をPerl以外の言語で実装しても構わない。プラグイン開発者に言語の選択が委ねられているのだ。

- inputs

  このメトリクスプラグインが、どのような入力値を必要としているか、その一覧を出力するターゲットである。これは必須ターゲットである。 プラグインディレクトリ内で、ユーザが「make inputs」と叩けば、次のように出力される。

  ```
  dividend divisor
  ```

  「dividend」と「divisor」の2入力を与えれば良い、と分かる。これは入力データのキーを機械的に出力するだけで良い。 SynapseScriptの処理系が、この情報を利用することになる。入力データのキーと、区切り文字としての半角スペースと改行のみが許されている。

- outputs

  このメトリクスプラグインが、どのような出力値を応答するか、その一覧を出力するターゲットである。これは必須ターゲットである。 プラグインディレクトリ内で、ユーザが「make outputs」と叩けば、次のように出力される。

  ```
  quotient floorQuotient remainder
  ```

  他の説明は、ターゲット「inputs」と同様である。

- help

  このメトリクスプラグインが、どのような変換を行うかの説明文を表示するターゲットである。これは任意ターゲットである。（ただし、利便性のため、できる限り用意していただきたい。） 上述したターゲット「inputs」「outputs」は機械的なものであったが、このターゲットでは、人が読むことを前提にした説明文を表示することになる。 プラグイン「Division2」における説明文を以下に示そう。

```
#
# Division2 - Metrics for SynapseScript
#
#  Created by M.Miyazaki [2013.01.06]
#  Modified by M.Miyazaki [2013.02.18]
#


### Description
  Calculate division.

### Inputs
  dividend : A number that divided.
  divisor  : A number that divide.

### Outputs
  quotient      : Quotient of division.
  floorQuotient : Quotient of division with floor.
  remainder     : Remainder of division.

### Examples
  Try this example...

  $ make dividend=`printf "100" | base64` divisor=`printf "21" | base64`
  > {"quotient" : "NC43NjE5MDQ3NjE5MDQ3Ng==", "floorQuotient" : "NA==", "remainder" : "MTY="}
  $ printf "NC43NjE5MDQ3NjE5MDQ3Ng==" | base64 --decode
  > 4.76190476190476
  $ printf "NA==" | base64 --decode
  > 4
  $ printf "MTY=" | base64 --decode
  > 16


Thank you for trying SynapseScript! :-)
```

  その記述内容や形式は基本的に自由であるが、1.何を実現するプラグインか、2.何を入力するか、3.何を出力するか、の3つについては記載していただきたい。 また、可能であれば利用例を掲載していただきたい。きっとユーザの利用を助けるだろう。

説明すべきターゲットは以上の4種類である。これら4つのターゲット名と重複しなければ、他のターゲットを用意していただいても構わない。

開発例

ここでは実例として、四則演算のメトリクスプラグイン「Arithmetic」を開発する。

手順1. プラグインを設計する

まずは、プラグインの設計を行う。プラグインは多入力多出力を許す関数であり、図5のように描くことができよう。 今回は、何かしらの2数を入力して、加算・減算・乗算・除算を行い、和・差・積・商を出力するものである。 つまり「Arithmetic」は、2入力4出力のメトリクスプラグインだ。

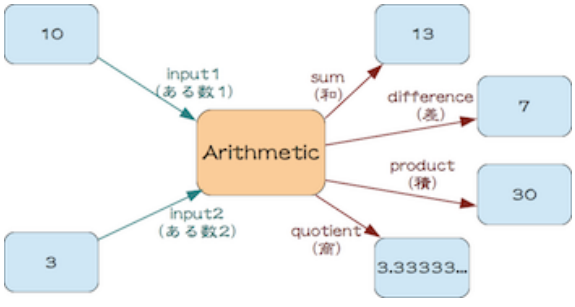


図5 プラグイン「Arithmetic」のメトリクスグラフ

手順2. プラグインの形を整える

次に、メトリクスプラグインの基盤となるファイル群を作成する。プラグインディレクトリ「Arithmetic」を作成し、その中に「Makefile」「help.txt」を作成する。 また、算術処理はPHPで記述することにして「Scripts/Arithmetic.php」を作成する。いずれも空のテキストファイルで良い。 もちろん、PHPでなく他の言語でも良い。また、Makefile内で計算処理を済ませてしまっても構わない。

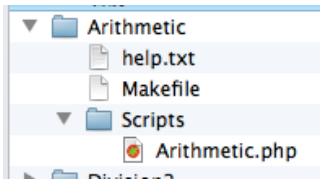


図6 プラグイン「Arithmetic」のファイル構成

手順3. Makefileの作成

メトリクスプラグインの核となる「Makefile」を作成する。以下のように記述すれば良いだろう。 実を言うとその中身は、プラグイン「Division2」のものと酷似している。それほど形式的なものなのだ。テンプレートを1つ用意しておけば、あとはそれを使い回すだけで良い。

図5で示したとおり、入力は「input1」「input2」の2つ、出力は「sum」「difference」「product」「quotient」の4つである。

```
#
# Arithmetic - Metrics for SynapseScript
#
#  Created by M.Miyazaki [2013.02.19]
#

all:
    @php ./Scripts/Arithmetic.php "$(input1)" "$(input2)"

inputs:
    @echo input1 input2

outputs:
    @echo sum difference product quotient

help:
    @less help.txt
```

手順4. 変換処理の実装

入力値を出力値に変換する部分を実装しよう。今回はPHPを用いて以下のように記述した。これを、Scriptsディレクトリ内のArithmetic.phpとして保存しておく。 説明用の簡単な実装を示しているが、本来は適切な例外処理を施すべきである。（非数値の入力やゼロ除算等を考慮し、対処すべきである。）

```
<?php
#
# Arithmetic - Metrics for SynapseScript
#
#  Created by M.Miyazaki [2013.02.19]
#

$input1 = base64_decode($_SERVER['argv'][1]);
$input2 = base64_decode($_SERVER['argv'][2]);

$sum        = base64_encode($input1 + $input2);
$difference = base64_encode($input1 - $input2);
$product    = base64_encode($input1 * $input2);
$quotient   = base64_encode($input1 / $input2);

$formatString = "{\"sum\" : \"%s\", \"difference\" : \"%s\", \"product\" : \"%s\", \"quotient\" : \"%s\"}\n";
printf($formatString, $sum, $difference, $product, $quotient);

?>
```

手順5. 動作確認

変換処理を実装したら、その動作確認を行う。先の図5で示したとおり「10」と「3」を入力して、その動作を検証する。

```
$ cd /path/to/Arithmetic

$ make input1=`printf "10" | base64` input2=`printf "3" | base64`
{"sum" : "MTM=", "difference" : "Nw==", "product" : "MzA=", "quotient" : "My4zMzMzMzMzMzMzMz"}

$ printf "MTM=" | base64 --decode
13

$ printf "Nw==" | base64 --decode
7

$ printf "MzA=" | base64 --decode
30

$ printf "My4zMzMzMzMzMzMzMz" | base64 --decode
3.3333333333333
```

手順6. 説明文の用意

「make help」で表示されるメトリクスプラグインの説明文を用意する。説明文を以下のように記述し、help.txtとして保存しておく。

```
#
# Arithmetic - Metrics for SynapseScript
#
#  Created by M.Miyazaki [2013.02.19]
#


### Description
  Do arithmetic operations.
  Addition, subtraction, multiplication, division.

### Inputs
  input1 : A number.
  input2 : A number.

### Outputs
  sum        : Result of addition.
  difference : Result of subtraction.
  product    : Result of multiplication.
  quotient   : Result of division.

### Examples
  Try this example...

  $ make input1=`printf "10" | base64` input2=`printf "3" | base64`
  > {"sum" : "MTM=", "difference" : "Nw==", "product" : "MzA=", "quotient" : "My4zMzMzMzMzMzMzMz"}
  $ printf "MTM=" | base64 --decode
  > 13
  $ printf "Nw==" | base64 --decode
  > 7
  $ printf "MzA=" | base64 --decode
  > 30
  $ printf "My4zMzMzMzMzMzMzMz" | base64 --decode
  > 3.3333333333333


Thank you for trying SynapseScript! :-)
```

手順7. 完成

以上の手順により、メトリクスプラグインを開発することができた。その成果物は、こちら（Arithmetic.zip）のようになる。

学生証番号：947424　氏名：宮崎 雅文

SynapseScript 言語マニュアル

はじめに

本稿は、メトリクスのグラフ構造を表現するためのデータ記述言語「SynapseScript」に関するマニュアルである。 仕様や具体例の提示によって、SynapseScript利用時の補助を行うことが目的である。

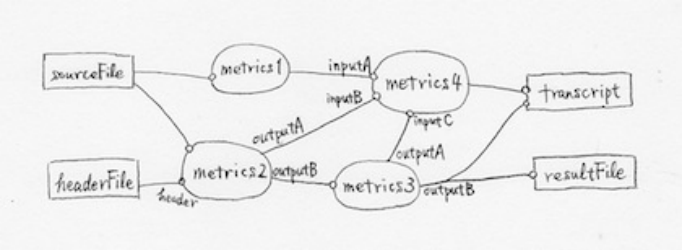


図1 メトリクスグラフの例

対象を何かしらの指標に基づいて計測する「メトリクス」は、図1のようなグラフ構造で表現することができる。 グラフのノードは、多入力多出力の関数に相当する「メトリクスプラグイン」として実現し、グラフのアークは、ノード同士の繋ぎ役として「SynapseScript」を提案するのである。

ちなみに「SynapseScript」という名称は、メトリクスグラフが神経細胞の接続構造（ニューラルネットワーク）に似ていたことを由来としたものであり、 その構造の要となる「接続」の仕組み・構造を意味する「シナプス」という言葉を採用したのだ。

定義

ここでは、本ガイドラインに記載されている用語の定義を行う。

- SynapseScript
  関数の役割を果たす「メトリクスプラグイン」を接続するための仕組み。データ記述言語。
- 入力（input）/ 出力（output）
  変換系（メトリクスプラグイン）がデータＡをデータＢに変換する時、変換前のデータ（つまりＡ）を入力と呼び、変換後のデータ（つまりＢ）を出力と呼ぶ。 また、この変換のため変換系に対してデータＡの提供を行うが、この行為自体も入力と呼び、変換系がデータＢを作り出す行為も出力と呼ぶ。
- プログラム
  ソフトウェアプログラム。コンピュータを操作するための言語表現。ソースコードとも。
- メトリクス
  プログラムなどの評価対象を、何らかの尺度・指標（ものさし）で計測することを言う。
- メトリクスプラグイン
  具体的なメトリクスの処理（写像処理）を行う機能単位。何らかの入力値に対応する出力値を応答する変換系（翻訳者）である。 例えば、プログラムを入力して、そのプログラムの行数を計測して出力するなどを実現するもの。

記法

概要 ― 具体例を添えて

具体的なメトリクスグラフの例を挙げ、その構造を表すSynapseScriptの記述（スクリプト）を紹介しよう。

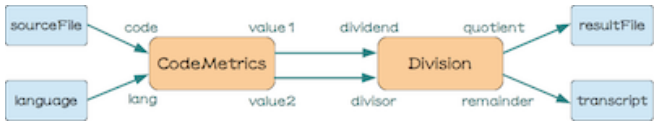


図2 サンプルのメトリクスグラフ

```
/*** Assignments ***/
sourceFile  = File("/path/to/Programs/program.py", "read");
language    = Value("Python");
codeMetrics = Metrics("/path/to/Plugins/CodeMetrics");
division    = Metrics("/path/to/Plugins/Division");
resultFile  = File("/path/to/Results/ResultFile.txt", "write");
transcript  = Transcript();

/*** Synapses ***/
sourceFile
    to: codeMetrics.code
    send: value1   to: division.dividend
    send: quotient to: resultFile;
language
    to: codeMetrics.lang
    send: value2    to: division.divisor
    send: remainder to: transcript;
```

スクリプトは「代入文」と「シナプス文」の2種類で構成される。それぞれを以下に詳説する。

代入文

```
/*** Assignments ***/
sourceFile  = File("/path/to/Programs/program.py", "read");
language    = Value("Python");
codeMetrics = Metrics("/path/to/Plugins/CodeMetrics");
division    = Metrics("/path/to/Plugins/Division");
resultFile  = File("/path/to/Results/ResultFile.txt", "write");
transcript  = Transcript();
```

代入文は、グラフのノードたちを定義するものである。等号「=」を挟み、右辺はノードのコンストラクタ、左辺は対応付ける変数を記述する。ここで命名した変数名は、後述のシナプス文でグラフ構造を表現する際に用いることになる。

単一の代入文は、次のように記述される。また、利用できるコンストラクタ名の例も示そう。

```
変数名 = コンストラクタ名 ( 引数群 );
```

コンストラクタ名	役割
Value	データのノードを生成
File	ファイルのノードを生成
Metrics	メトリクスプラグインのノードを生成
Transcript	トランスクリプトのノードを生成

- コンストラクタ「Value」

  データ（値）のノードを生成する。具体的なデータを引数に持たせる。生成されたノードは出力のみ行う。 以下に示すのは、「あいうえお」という文字列をデータとするノードを生成し、変数「stringValue」に束縛する場合の例である。

  ```
  stringValue = Value("あいうえお");
  ```

  引数として渡すデータは、記述できるものであれば数値でも文字列でも何でも良い。ただし、二重引用符「"」で囲む必要がある。 もし、デ

ータそのものに二重引用符が含まれる場合（「あい"う"えお」など）は、二重引用符を連続して記述しなければならない。つまり、次のような記述になる。

```
stringValue = Value("あい""う""えお");
```

- コンストラクタ「File」
  ファイルのノードを生成する。取り扱うファイルのパスを引数に持たせる。また、第二引数として「read」「write」「append」を指定する。「read」の指定とともに生成されたノードは出力（データの読み出し）のみを行い、「write」「append」の指定とともに生成されたノードは入力（データの書き込み・追記）のみを行う。 以下に示すのは、「/path/to/hoge.txt」というファイルからデータを読み出すノードを生成する例である。

```
hogeFile = File("/path/to/hoge.txt", "read");
```

- コンストラクタ「Metrics」
  メトリクスプラグインのノードを生成する。対象となるメトリクスプラグインのパスを引数に持たせる。生成されたノードは入力及び出力を行う。 以下に示すのは、「/path/to/Plugins/Foobar」というプラグインに対応するノードを生成する例である。

```
foobar = Metrics("/path/to/Plugins/Foobar");
```

- コンストラクタ「Transcript」
  トランスクリプト（文字列を表示する端末）のノードを生成する。生成されたノードは出力のみ行う。生成例を以下に示す。

```
transcript = Transcript();
```

以上、4種類のコンストラクタを示したが、処理系によっては「Transcript」を用意できない場合もあろう。つまり、これらは処理系に依存することになる。 また、この4種類以外のコンストラクタを用意することもあり得る。

シナプス文

```
/*** Synapses ***/
sourceFile
    to: codeMetrics.code
    send: value1   to: division.dividend
    send: quotient to: resultFile;
language
    to: codeMetrics.lang
    send: value2    to: division.divisor
    send: remainder to: transcript;
```

シナプス文は、代入文で定義したノードたちの繋ぎ方を明記するものである。「あるノードの出力」と「あるノードの入力」を繋げるために、to:文（またはsend:to:文）を記述することになる。

- ノード同士の接続
  あるノード「nodeA」の出力を、あるノード「nodeB」の入力に繋げる場合、to:文を用いて次のように記述する。

```
nodeA to: nodeB;
```

ちなみに、これはラベル名の記述を（特別に）省略しているに過ぎない。デフォルトの出力ラベル名「output」入力ラベル名「input」を用いて、次のように記述することと相等である。

```
nodeA.output to: nodeB.input;
```

では、ラベル名が別に定められている場合について。あるノード「nodeA」の出力（ラベル「output1」）を、あるノード「nodeB」の入力（ラベル「input1」）に繋げる場合、次のように記述する。

```
nodeA.output1 to: nodeB.input1;
```

これは、send:to:文を用いて次のように記述することもできる。意味は全く同じである。

```
nodeA send: output1 to: nodeB.input1;
```

- 連続した接続

あるノード「nodeA」の出力を、あるノード「nodeB」の入力とし、また「nodeB」の出力を、あるノード「nodeC」の入力とする場合、次のように記述する。

```
nodeA to: nodeB;
nodeB to: nodeC;
```

この連続した接続を、以下のように簡便な表現に直すこともできる。

```
nodeA to: nodeB to: nodeC;
```

ラベルが指定されている場合も同様に、連続した接続を表現することができる。ちなみに、インデントの形式は自由である。

```
nodeA
    send: output1 to: nodeB.inputA
    send: outputA to: nodeC.input1;
```

補遺

字句解析向けの正規表現

字句解析器生成系の一「flex」向けに拵えた正規表現を掲載する。PDF形式の正規表現（rexp.lex.pdf）も用意している。

```
%{
#include "defs.h"
%}
%option nounput
%%
"send:"                         { return(SEND); }
"to:"                           { return(TO); }
[_a-zA-Z][_a-zA-Z0-9]*          { return(ID); }
"="                             { return(GETS); }
";"                             { return(SEMICOLON); }
"\."                            { return(PERIOD); }
"("                             { return(LPAR); }
")"                             { return(RPAR); }
","                             { return(COMMA); }
\"([^\"]|\"\")*\"               { return(STRING); }
"\r\n"|"\r"|"\n"|" "|"\t"       { }
"/*"([^*]|"*"+[^*/])*"*"+"/"    { return(COMMENT); }
.                               { return(UNKNOWN); }
%%
int yywrap(void) {
  return(1);
}
```

構文解析向けのBNF

構文解析器生成系の一「Yacc」向けに拵えたBNF（バッカス・ナウア記法）を掲載する。 PDF形式のBNF（BNF.pdf）と抽象構文木の生成を行う意味解析付きのBNF（syns.yac.pdf）も用意している。

```
%{
#include "defs.h"
#define YYSTYPE char *
char s[1024];
%}
%token  SEND TO GETS SEMICOLON PERIOD LPAR RPAR COMMA ID STRING COMMENT UNKNOWN
%%
Program
    : Statements                            { }
```

```
Statements
    :                                       { }
    | Statement                             { }
    | Statements Statement                  { }
Statement
    : Assignment SEMICOLON                  { }
    | Synapse SEMICOLON                     { }
    | Comment                               { }
Assignment
    : VariableName GETS Constructor         { }
Synapse
    : Variable Mappings                     { }
Mappings
    : Mapping                               { }
    | Mappings Mapping                      { }
Mapping
    : TO Variable                           { }
    | SEND MemberName TO Variable           { }
Variable
    : VariableName                          { }
    | VariableName PERIOD MemberName        { }
VariableName
    : ID                                    { }
MemberName
    : ID                                    { }
Constructor
    : ConstructorName LPAR RPAR             { }
    | ConstructorName LPAR Arguments RPAR   { }
ConstructorName
    : ID                                    { }
Arguments
    : Argument                              { }
    | Arguments COMMA Argument              { }
Argument
    : STRING                                { }
Comment
    : COMMENT                               { }
%%
#include "lex.yy.c"
void yyerror(char *s) {
    fprintf(stderr, "\n%s at %d: nearby \"%s\"\n\n", s, linecounter, yytext);
    exit(EXIT_FAILURE);
}
```

学生証番号：947424　氏名：宮崎 雅文

マニュアル（MetricsTool.zipの取り扱い）

動作環境

ここで紹介するメトリクスツールの動作には、以下の構成要素が必要となる。

- VisualWorks (7.8 with じゅん for Smalltalk)
- make (GNU Make 3.81)
- gcc (4.2.1)
- awk (20070501)
- javac, java (1.7.0_09)
- Perl (5.12.4)
- PHP (5.3.15)
- Python (2.7.2)
- ruby (1.8.7)
- sh (GNU bash 3.2.48)

同梱物

- MetricsTool.zip

必要なファイルは、上記アーカイブ（MetricsTool.zip）にまとめている。その内容は以下の通りである。

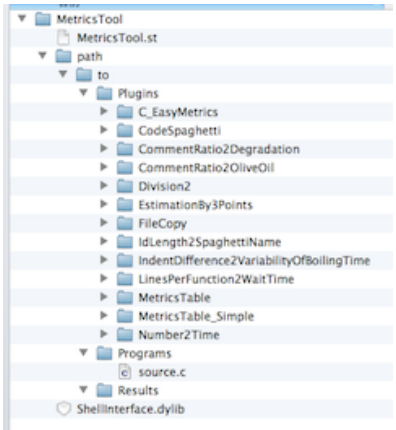


図1 アーカイブの内容

- ./MetricsTool.st

  Smalltalkで実装されたSynapseScript処理系のプログラム。VisualWorksで読み込むもの。

- ./path/to/Plugins

  メトリクスプラグインが収められているディレクトリ。13種類のプラグインを同梱している。

- ./path/to/Programs
  サンプルのC言語プログラムを同梱している。スパゲッティへの可食化の例題プログラムのために用意した。
- ./path/to/Results
  スパゲッティへの可食化の例題プログラムを動作させた際にレシピが出力されるディレクトリ。初期状態は空。
- ./ShellInterface.dylib
  Smalltalk（VisualWorks）からシェルを利用する際に必要となるライブラリファイル。

また、その動作にはSmalltalkの処理系であるVisualWorks（じゅん for Smalltalkを含むもの）が必要になる。外部サイトより入手されたい。

マニュアル１（例題プログラム動作までの手順）

ここでは、同梱物に含まれている例題プログラム（スパゲッティへの可食化の例題）を動作させる手順を説明する。

手順1. 同梱アーカイブ（MetricsTool.zip）の展開と移動

上に用意したアーカイブ（MetricsTool.zip）を任意の場所にダウンロードし、展開する。 その内容物のうち「path」と「ShellInterface.dylib」を、図2に示すように、VisualWorksの仮想マシンが存在するディレクトリに移動する。

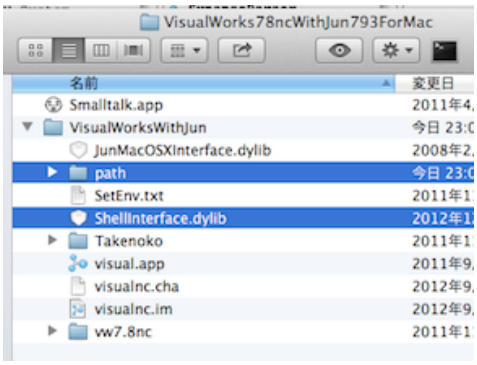


図2 仮想マシンのディレクトリ構成

手順3. VisualWorksを立ち上げて、stファイルを読み込む

VisualWorksを起動し、ファイルブラウザから「MetricsTool.st」をファイルインする。

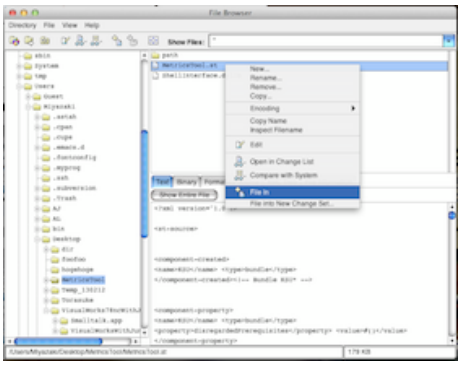


図3 MetricsTool.stをファイルイン

手順4. 可食化の例題プログラムを実行する

システムブラウザ（リファクタリングブラウザ）を開き、バンドル「KSU」下のパッケージ「KSU-Synapse」に属する、クラス「KSU.SynapseInterpreter」のクラスメッセージ「exampleCodeSpaghettiForStartup」を実行する。 または、ワークスペース等で、以下のフレーズを実行されたい。

```
KSU.SynapseInterpreter exampleCodeSpaghettiForStartup
```

実行と同時に、サンプルプログラム「./path/to/Programs/source.c」をスパゲッティに可食化する。処理を終えると、スパゲッティのレシピが表示される。 もし、しばらく経っても表示されない場合は、「./path/to/Results」ディレクトリを参照されたい。HTMLファイルのレシピが出力さ

れているはずである。

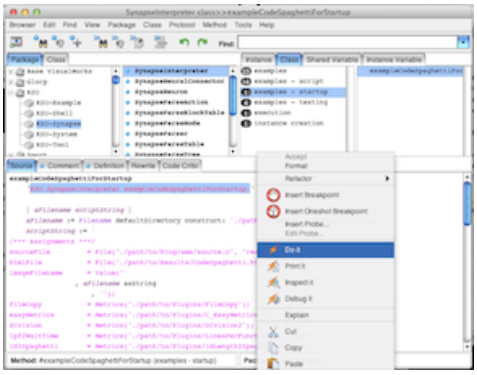


図4 スパゲッティへの可食化の例題プログラム

図5 スパゲッティのレシピ

マニュアル２（自身でSynapseScriptを記述して実行する際の手順）

ここでは、ユーザ自身がSynapseScriptを記述して、その実行を行う場合の例を示す。マニュアル１の手順2までを済ませているものとして説明を進める。 また、例題として割り算のメトリクスプラグイン（Division2）を用いる。

手順3. ワークスペース上で実行の準備を行う

一枚のワークスペースを用意し、以下のように記述する。

```
KSU.SynapseInterpreter script: ''
```

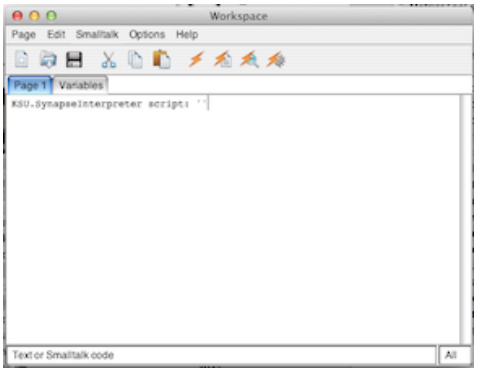


図6 ワークスペースでの準備

手順4. SynapseScriptを記述する

準備したプログラムフレーズの引用符内にSynapseScriptを記述する。 ここでは「20÷3」といった割り算の処理を行い、その計算結果（商）をトランスクリプトに表示することとしよう。SynapseScriptの記述は以下の通り。

```
/*** Assignments ***/
value1 = Value("20");
value2 = Value("3");
division = Metrics("./path/to/Plugins/Division2");
transcript = Transcript();

/*** Synapses ***/
value1 to: division.dividend;
value2 to: division.divisor;
division send: quotient to: transcript;
```

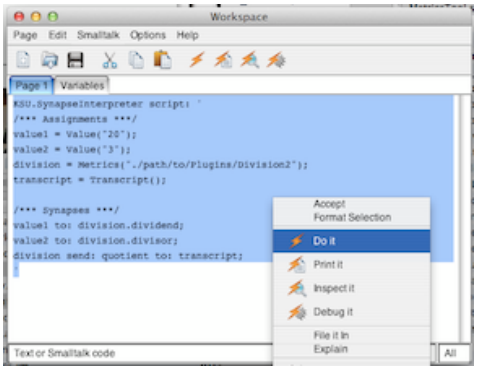


図7 割り算のSynapseScript

手順5. 例題プログラムを実行する

ワークスペースに記述したプログラムを全て選択して実行してみると、確かにトランスクリプト上に「20÷3」の解として「6.66666...」が表示されている。

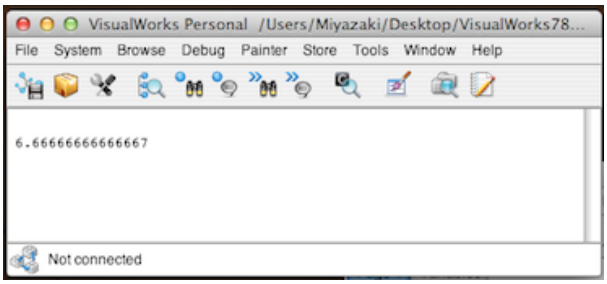


図8 トランスクリプト上に結果表示

学生証番号：947424　氏名：宮崎 雅文

実施日：西暦　　年　　月　　日　学生証番号：

スパゲッティの試食評価実験 アンケート用紙

1. 概要

本実験は、京都産業大学「コンピュータ理工学 特別研究ⅡA・B」への取り組み（以下「研究」）のため実施するものです。被験者の皆様には、スパゲッティの試食・評価、ならびに研究に関連するアンケートへの記入に取り組んでいただきます。

2. 注意事項

本実験の実施に際して、以下に記載します注意事項をご理解いただき、また指定する事柄を遵守してください。実験の妥当性を確保するため皆様のご協力をお願い致します。

- アンケート用紙上に記載されている説明文をよく理解した上で、アンケートに回答してください。
- 実験開始から実験終了まで、この実験に関する話題を他の被験者との間で交わさないでください。
- 実験中に不明点が生じた場合は「他の被験者」ではなく「実施者」に対して質問してください。
- 説明文の内容に関わらず、実施者から別途の指示があった場合は、その指示に従ってください。

3. 手順

事前説明 → 実験開始 → 試食とアンケート記入(計４回) → アンケート(総括) → 実験終了

4. 試食とアンケート記入

お出ししたそれぞれのスパゲッティを食した上で、以下の設問にご回答ください。

【Q1】召し上がったスパゲッティ（１回目）に対して、直感的にどのような印象を持ちましたか。
以下に並べる７つの評価のうち１つを選び、丸で囲んでください。

とても良い ・ 良い ・ 少し良い ・ どちらとも言えない ・ 少し悪い ・ 悪い ・ とても悪い

【Q2】Q1のように感じた理由を述べるとすれば、どのような理由が挙げられますか。

学生証番号：＿＿＿＿＿＿＿＿

スパゲッティの試食評価実験 アンケート用紙

これ以降お出しするスパゲッティは、すべて、１回目のスパゲッティを基準に評価してください。
また、一度書き終えた設問に遡って修正することは控えてください。
Q3・Q5・Q7 では、用紙下部の語群から１つ選択しますが、選択が重複しても問題ありません。

【Q3】スパゲッティ（２回目）で最も特徴的だった指標を語群から選び、その理由をお書きください。

指標：＿＿＿＿＿＿＿＿

理由：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

【Q4】Q3 のスパゲッティに対して、直感的にどのような印象を持ちましたか。

とても良い ・ 良い ・ 少し良い ・ どちらとも言えない ・ 少し悪い ・ 悪い ・ とても悪い

【Q5】スパゲッティ（３回目）で最も特徴的だった指標を語群から選び、その理由をお書きください。

指標：＿＿＿＿＿＿＿＿

理由：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

【Q6】Q5 のスパゲッティに対して、直感的にどのような印象を持ちましたか。

とても良い ・ 良い ・ 少し良い ・ どちらとも言えない ・ 少し悪い ・ 悪い ・ とても悪い

【Q7】スパゲッティ（４回目）で最も特徴的だった指標を語群から選び、その理由をお書きください。

指標：＿＿＿＿＿＿＿＿

理由：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

【Q8】Q7 のスパゲッティに対して、直感的にどのような印象を持ちましたか。

とても良い ・ 良い ・ 少し良い ・ どちらとも言えない ・ 少し悪い ・ 悪い ・ とても悪い

語群

ゆで上げ後の待ち時間　　オリーブオイル量　　スパゲッティ料理名　　ゆで具合のばらつき

学生証番号：＿＿＿＿＿＿＿＿

スパゲッティの試食評価実験 アンケート用紙

5. スパゲッティ料理名

１回目の料理は「トマトソースのスパゲッティ」という名称でしたが、その点を踏まえた上で、以下の名称をどのように感じますか。

【Q.9】スパＡ

とても短い ・ 短い ・ やや短い ・ どちらとも言えない ・ やや長い ・ 長い ・ とても長い

印象：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

【Q.10】トマトソースのスパゲッティ

とても短い ・ 短い ・ やや短い ・ どちらとも言えない ・ やや長い ・ 長い ・ とても長い

印象：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

【Q.11】賀茂茄子と香草のスパゲッティ アンデス風 サルサ・ディ・ポモドーロを添えて

とても短い ・ 短い ・ やや短い ・ どちらとも言えない ・ やや長い ・ 長い ・ とても長い

印象：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

6. メトリクス結果の一覧表示に関する実験

実施者により提示される２種類のソフトウェアプログラムのメトリクス結果（一覧表示型）を見て、それぞれが「良いプログラム」か「悪いプログラム」かを判定し、判定結果とその理由を書いてください。

ただし、「プログラミング初学者のプログラムにおける良し悪し」に限定して判定していただきます。以下に挙げる４つの条件を満たすプログラムが「良いプログラム」です。条件を満たさない項目があれば「悪いプログラム」となります。

- インデント（字下げ）が整っていること。
- 適切な長さの識別子（変数名・関数名）を用いていること。
- プログラム中に一定量のコメントが付されていること。
- 機能ごとに関数化が施されていること。

【Q.12】１つ目

良いプログラム ・ どちらとも言えない ・ 悪いプログラム

理由：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

【Q.13】２つ目

良いプログラム ・ どちらとも言えない ・ 悪いプログラム

理由：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

学生証番号：＿＿＿＿＿＿＿＿

スパゲッティの試食評価実験 アンケート用紙

7. 「可視化」と「可食化」

本実験では、ソフトウェアメトリクス結果の可視化（数値を一覧表示するもの）と可食化（数値をスパゲッティに変換したもの）の例をご覧に入れましたが、最後に質問です。

【Q.14】ソフトウェアプログラムの問題点や改善点を発見する上で、メトリクス結果の可視化および可食化はそれぞれ、どれくらい「直感的な理解・認知」を得られる手法でしょうか。それぞれの手法を十点満点（0～10 の数値）で評価してください。

可視化（一覧化）：＿＿＿＿＿＿点

可食化（スパゲッティ化）：＿＿＿＿＿＿点

【Q.15】上記の点数で評価した理由をお書きください。

8. その他

本研究・本実験に関する、ご意見・ご感想・ご質問・ご叱責等がございましたら、ご記入ください。

長時間に渡るご協力、本当にありがとうございました。

索引

学生証番号：947424　氏名：宮崎 雅文