ソフトウェア工学の方法論と理論
(SOFTWARE ENGINEERING METHOD AND THEORY – A VISION STATEMENT)

By Ivar, Bertrand and Richard
(日本語翻訳[1] ： 平鍋健児)

1. 目的とスコープ(Purpose and scope)

Semat の最初の Call for Action では、今後 Semat initiative が扱おうとしている問題の概要を定義する。

Call for Action

ソフトウェア工学(Software Engineering)は未成熟なプラクティス(immature practices)によって、重大な阻害(gravely hampered)を今日受けている。例えば、具体的には以下のように:

- 言葉の流行が、工学の一分野というよりファッション業界のようだ。
- しっかりした広く受け入れられた、理論的基礎の欠如。
- 非常に多くの方法論(methods)とその派生。またそれらの違いがほとんど理解されずに作為的に強調されている。
- 信頼できる実験的評価(experimental evaluation)と妥当性確認(validation)の欠如。
- 産業界の実践(industry practice)と学界の研究(academic research)の乖離。

私たちは、ソフトウェア工学を堅固な理論および検証された原則とベストプラクティスを基礎として、再建するプロセスを支援する。そのプロセスは、以下の特徴を備えている。

- 広く合意された要素からなる、特定用途に拡張可能なカーネルを含み、
- 技術の問題と人の問題の両方を扱い、
- 産業界、学界、研究者そして、ユーザに支援され、
- 要求とテクノロジの変化に応じて追随できるような拡張性を備えている。

適切なソリューションを開発するためには、さらに詳細なフレームワークが必要だ。このビジョン・ステートメントはそのフレームワークであり、初年度に期待するゴールを含む、ソフトウェアにおける「要求文書」や「ロードマップ」と似たものである。

この活動への参加者は、ソフトウェアという職業の多くの立場からなる。例えば、現場のプログラマ、プロジェ

[1]訳注： この文書(日本語訳)は Semat に許可を得て平鍋が翻訳しているが Semat にレビューを得たものではない。正式な内容については元資料、http://www.semat.org/に公開された、http://www.semat.org/pub/Main/WebHome/SEMAT-vision.pdf2010 を参照のこと。この日本語訳は、2010/2/10 のバージョンを元にしている。

クトマネジャ、コンサルタント、コンピュータサイエンティストなどを含む。私たちはこれから、密度の濃い議論が起こるのを期待しており、このビジョン・ステートメントは特定のソリューションを押し付ける意図ではい。しかし、その議論がうまく機能するためには、このグループはビジョンを共有し、共通のゴールに合意し、最初のマイルストーンを定義し、そしてそのゴールに近づく中で得られる原則を受け入れる必要がある。このドキュメントは、ビジョン(第2節)、ゴール(第3節)、ルール(第4節)、原則(第5節)、1年間のマイルストーン(第6節)をそれぞれ定義しようとする試みだ。第3節の5つの主要なゴールを十分に理解するために、付録を特別に追加して、そのそれぞれについて掘り下げた。

これらの「長期ゴール」と第6節の「1年間のマイルストーン」は、それぞれお互いに必要な要素である: Semat の目標である根本的な変化には、数年かかるだろう。しかし、ムーブメントを起こすには、目に見える最初の結果を早期に達成することが必要だ。

2. ビジョン(The vision)

Semat のビジョンは、2段構成である。

- 付録も含めてこの文書の以下の章に書かれたすべてのゴールを達成すること。
- 人々が、自分たちの現在、未来のプラクティス(practices)、パターン(patterns)、メソッド(methods)を記述(describe)し、それらを組合せ(compose)、模倣(simulate)、応用(apply)、比較(compare)、評価(evaluate)、計測(measure)、教育(teach)、研究(research)できるようなプラットフォーム(カーネル)を作る。

3. カーネル(The kernel)

Call for Action は「広く合意された要素からなる、特定用途に拡張可能なカーネル」の合意へ私たちを導く。効果的なカーネルであるために、具体的で、焦点が絞られていて、かつ、小さくなければならない。

このカーネルのスコープは、この節に記述された要素に限られている。特に、カーネルは「統一方法論」への試みではない。

このカーネル活動の焦点は、すべてのソフトウェア工学活動において本質的な要素を特定して記述することである。

カーネルがカバーする領域として典型的な例は、チームワーク、プロジェクトマネジメント、プロセス改善などである。カーネルは他のエンジニアリング分野からのコンセプトをも統合するであろう。

カーネルは、変化を受け入れなければならない。最初の 12 ヶ月では、今日のソフトウェアづくりで使われている意味のあるメソッド(methods)、プラクティス(practices)、パターン(patterns)(図 1)を全部把握し、かつ、将来のこの領域の進化に適用できるようなカーネルを特定する。

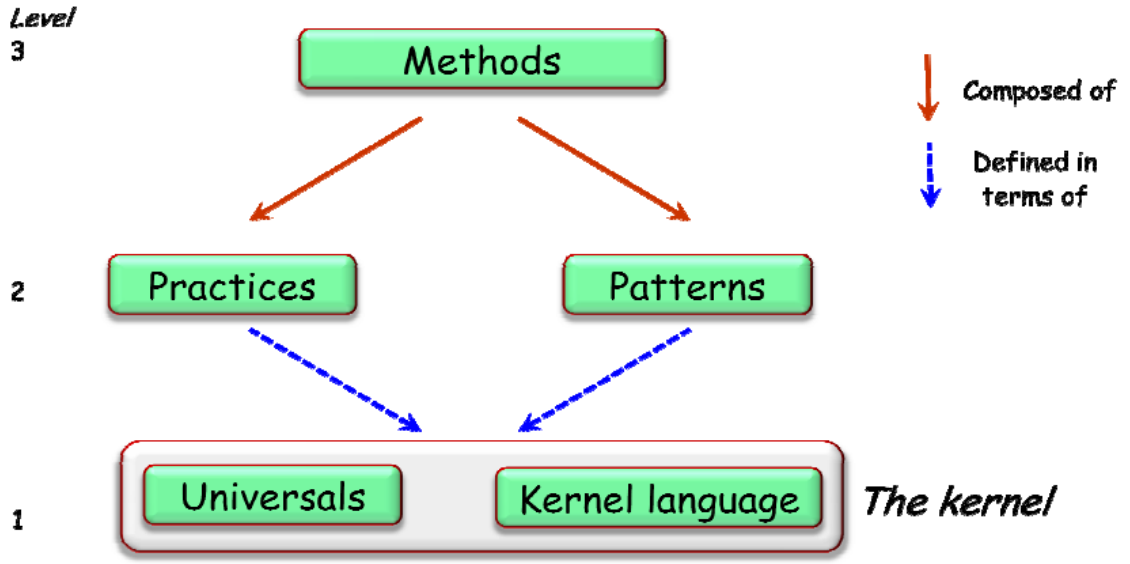

図 1 The Semat Diamond

このカーネルは、ソフトウェア開発の活動(acts)と成果物(artifacts)の具体的表現を含み、幅広いソフトウェアプロジェクトに適用可能でなければならない。さらに特定の方法論、プラクティス、パターンに適応する拡張言語を提供する必要がある。

これらのニーズを満たすために、カーネルは、以下の3つの主要素を必要とする。

- カーネル(kernel):ユニバーサル（汎用要素: universal)とカーネル言語(kernel language)(図 1 の Level 1)
- プラクティス(practices)とパターン(patterns):カーネル言語によって定義される(Level 2)
- メソッド(methods):プラクティスとパターンの組み合わせによって定義される(Level 3)

「パターン」についてはここでの定義を説明する必要があるだろう。このドキュメントでは、複数のプラクティスに渡って適用される汎用的な活動を表現するのに、この言葉を使っている。例えば、多くのプラクティスではワークショップを行うし、自己組織化したチームに依存している。このようなコンセプトをパターンとして記述することで、個々のプラクティスでそれらの詳細な記述を繰り返すのを避ける。

4. ゴール(The goals)

この活動は5つの別トラックで進められる。

- **定義(Definitions)**: ソフトウェア工学、およびその領域のその他本質的なコンセプトを定義する。
- **理論(Theory)**: 本質的な助力を提供する理論(特に数学からの)。
- **ユニバーサル(Universals)**: Semat カーネルに組み込むべき、ソフトウェア工学の汎用要素を特定する。
- **カーネル言語(Kernel language)**: ユニバーサル、プラクティス、パターンを記述する言語を定義する。
- **評価(Assessment)**: ソフトウェア工学のプラクティス、理論を評価する手法。(Semat 自体の評価も含む)

すべてのトラックは、12 ヶ月以内に目に見える結果を出すべきである。しかし、結果の範囲は、大きく異なるだろう。もっとも短期の結果が必要とされるのは、トラック 3 と 4 である。なぜなら、全体の基礎となるカーネル要素を特定して、この活動全体への基礎を提供し、ソフトウェア産業でのメソッドの混乱を抑え、さらにソフトウェア工学教育を改善する必要があるからである。

付録ではそれぞれのトラックを開始するためのアイディアを提案している。

5. 原則(The principles)

Semat の運営方法(modus operandi)が活動に先立って定められるが、いくつかの一般原則はその成功に本質的だ。原則 1 から 9 は、「カーネル」と呼ばれる Semat の最終成果に適用される。10 から 13 の成果を得るプロセスに適用される。

1. **品質(Quality)**。主目標はソフトウェアプロダクトとプロセスの改善にある。
2. **シンプルさ(Simplicity)**。カーネルには本質的なコンセプトのみを含む。
3. **理論(Theory)**。カーネルは堅固で厳密な理論的な基礎の上に築かれる。
4. **現実性とスケーラビリティ(Realism and scalability)**。カーネルは実践的なプロジェクト(大規模プロジェクトを含む)に適用可能で、そこで検証可能な手法でなければならない。
5. **正当性(Justification)**。すべての提案は、明確な論拠によって正当化されなければならない。
6. **反証可能性(Falsifiability)**。すべての主張は、実験的な評価と反論を受けなければならない。
7. **先見性(Forward-looking perspective)**。前世代に起こった方法論の淘汰を考慮に入れつつも、完全な互換性には縛られない。
8. **モジュール性(Modularity)**。プラクティスとパターンはカーネルを使って定義され、それぞれの組織の

ニーズに合うように組み合わせたり調整したりできる。

9. **自己改善(Self-improvement)**。カーネルは自身の進化を可能にする仕組みを搭載しなければならない。
10. **オープン性(Openness)**。カーネルの開発においては、Semat 活動のメンバーからの適切な形式の示唆は、すべて考慮対象とされなければならない。
11. **公平性(Fairness)**。貢献するすべてのアイディアは、功績として評価されなければならない。どんな側面も、特定のステークホルダやコミュニティの利益に偏って設計されてはならない。
12. **目的性(Objectivity)**。アイディアは、前もって明確に定義された目的性の判断基準によって評価されなければならない。
13. **タイムリー性(Timeliness)**。進捗と結果をデリバリするために、締め切りを設けてそれを監視しなければならない。

6. 一年間のマイルストーン(One-year milestones)

このプロジェクト開始から1年後に期待する成果は以下である。それぞれが、「ゴール」として特定された5つの方向のうちの1つと対応しており、そのゴールに対しての最初の客観的評価可能な進捗である。

1. 「ソフトウェア工学」(Software Engineering)の定義、およびプラクティスとパターンに必要な基礎コンセプトの定義を含む、1セットの定義。
2. Semat のポテンシャルを収容することができ、特定のソフトウェア工学プラクティスへの成功適用例によって立証された、特定の理論や理論分野の識別。
3. 重要なメソッドで使われたいくつかの特定のプラクティス(少なくとも1つは汎用要素以外の開発で使われたもの)において妥当性検証できる、1セットの汎用要素。
4. カーネル言語の定義と汎用要素がそれによってうまく記述できること。
5. ソフトウェアプラクティス、プロダクト、ピープルを評価するに十分であり、かつ、いくつかのプロジェクトへの適用に成功実績によって裏打ちされた、1セットのメトリクス。

付録(Appendices)

５つのビジョン・ステートメントのゴールそれぞれを、５つの付録によって詳細化する： すなわち、「定義」、「理論」、「汎用要素」、「カーネル言語」、「アセスメント」である。

付録 1:　　定義(Definitions)

一般に、多くの技術的議論は、内容への反論ではなく用語の問題を含んでいる。「定義」トラックは、プロセスの初期に必要な用語の定義の合意を得ることを意図しており、他のトラックから発生する用語定義のトラッキングとカテゴライズを行う。

A1.1 ソフトウェアエ学とは何か？

定義において、1つのよい出発点であり代表例は、「ソフトウェアエ学」(software engineering)そのものの定義であろう。明白な定義(「ソフトウェアエ学は工学の手法(methods)／学問領域(discipline)をソフトウェア分野(field of software)へ適用したもの(application)である」)は、「工学」が一般的にうまく定義できていないため、もしくは、ソフトウェアへの適用がどう定義に影響するかの記述がないため、ある人たちにとっては不十分であると感じられる。別の極端な例として、Wikipedia には SWEBOK(the Software Engineering Body of Knowledge)から借りてきた長い定義が載せてある(「ソフトウェアエ学は、ソフトウェアの開発(development)、オペレーション(operation)、保守(maintenance)に対してのシステマティック(systematic)で、系統だった(disciplined)、定量的な(quantifiable)アプローチの適用、およびこういったアプローチの研究である。すなわち、ソフトウェアへの工学の適用である。」)。この定義にはがっかりする。形容詞の列は冗長であり(disciplined は systematic とそんなに変わらない)、恣意的(なぜ、オペレーションが、例えばドキュメンテーション(documentation)でなく選ばれたか？）、そして、最後の「すなわち」は、その前にあるすべての文言を無意味にしてしまいかねない。多くのディスカッションがすでに Semat ブログでこのソフトウェアエ学の定義のトピックで交わされた。各ポジションをサマリはすると、以下のようになる。

- ソフトウェアエ学は、正式な学位(formal study)、経験(experience)、敬意(respect)、創造性(creativity)、規律(discipline)を必要とする学問領域である。(皮肉がこもった以下を参照 http://parijatmishra.wordpress.com/2010/01/08/188/)
- 上記で引用した Wikipedia で使われている SWEBOK の定義。
- American Engineers' Council for Professional Development の定義。そこでは***工学***を、構造(structure)、機械(machines)、器具(apparatus)、生産プロセス(manufacturing process)、およびそれらを単独もしくは組み合わせて利用する活動を設計・開発(to design or develop)するために科学的原理(scientific principles)を創造的に(creatively)適用すること。あるいは、上記同様のものの組み立て(to construct)やオペレーション(to operate)へ、設計を認識したうえで適用すること。あるいは、それらの振る舞い(behavior)を特定の条件の下で予測する(to forecast)活動へ適用すること。意図する機能、オペレーションの経済性、生命と財産への安全性に関するすべて。(http://www.sciencedaily.com/articles/e/engineering.htm 参照。"works"

を”software works”に置き換えて software engineering が定義できるかもしれない）。

- Semat 名者の一人は上記定義には批判的であり、ソフトウェアエ学は、工芸(craft)、協調ゲーム(cooperative gaming)、リーンプロセス(lean processes)、「知識獲得としての設計」(design as knowledge acquisition) の要素を含むべきだと主張している。(http://alistair.cockburn.us/The+end+of+software+engineering+and+the+start+of+economic-cooperative+gaming を参照)。

おそらくこれらの定義は、それぞれそんなにかけ離れたものではない。どれも「科学的知識と原理をソフトウェアに適用する」という一般的な考え方を共有している。違った教育、知見、経験を備えた多くの専門家を集めて1つの共通の核心について合意することは、Semat Initiative のゴールの1つである。

A1.2 基礎的な定義を超えて

「ソフトウェアエ学」そのものの定義(あるいはその第一バージョン)は、少数の基礎的な他のコンセプトの定義とともに、最初の12ヶ月のゴール(第6節)の一部である。これらの用語は出発点にすぎず、他の Semat の活動(特にカーネルの定義)が、Semat コンポーネント(メソッド、プラクティス、パターン、ユニバーサル、カーネル言語)に含まれるコンセプトのすべてについて、定義の最終的採用を待つのは現実的ではない。定義プロセスは継続的、反復的であり、Semat の定義リポジトリを拡張し洗練していく。

付録 2:　理論(Theory)

工学の定義のひとつは、「その成果の構築は、科学的原則(最終的には数学)に則る」、ということである。ソフトウェアエ学もその例外ではない。強固な数学的基礎は、Semat 活動にとって本質的である。

ソフトウェアエ学が利用できる理論的基礎は数多くある。そのうちいくつかは既存の数学理論であり、そのまま広く適用することができる。例えばソフトウェアエ学の多くの領域で利用できるものとして、「確率」、「統計」、「待ち行列理論」がある(著名な研究者たちは、これまで何度もこうした理論をもっと広く使うべきだと主張してきた)。別の例として「カテゴリ理論」がある。これは、抽象データ型とオブジェクト指向プログラミングの発想の基礎となっている(多くの OO 開発者が知っているわけではないが)。別のケースでは、ソフトウェアの応用が新しい理論の開発や既存理論の展開を促進した例もある。例えば論理学は過去数十年間、プログラミング言語、オートマトン理論、新しく開発されたモデルチェッキング、抽象解釈(abstract interpretation)からのニーズが大きな原動力になっている。

ソフトウェアエ学は、技術だけの問題ではない。組織、マネジメント、コミュニケーション、協調、その他ソフトウェアの人間的な分野にもおよび、そこでは経済学、社会学、心理学といった学問分野から洞察が得られる。ソフトウェアエ学では、メソッド、プラクティス、パターン、ユニバーサルを開発するにあたってこれらの分

野からの理論も必要としている。

ソフトウェアにおいて、実践者と学界との溝は他の工学分野よりも深刻である。例えば電気エンジニアがマックスウェル方程式をとりあげて、実践と無関係な純粋に学問的な研究だ、などと言ったりすることは想像し難い。しかし、ソフトウェアのエンジニアとマネジャーが、現場にとっても重要な理論的貢献(例えば形式検証技術)に関して、そのようなコメントをすることはよくある。さらには、実践者がそういった手法のことをまったく聞いたこともない、というケースさえある。

アカデミック側にも、まったく罪がないわけではない。研究のいくつかは、科学的に挑戦的な課題であっても産業界の関心と関連付けることが難しいものもある。

2つのキャンプの交流はここ数年で改善している。形式手法を例にとると IT の数多くの分野で利用例が増えてきている(一見形式手法に見えないやり方で使われている例もある)。

ソフトウェア工学が他の工学分野のレベルにまで成長し、かつ、学界と産業界の溝を完全には取り除かないように(研究と応用は別の役割であるから望ましいことではない)しながら、理論と実践の健全な関係を構築することが、Semat のゴールの1つである。

「理論ワーキンググループ」のミッションは、このゴールをさらに推進することである。特に最初の段階では、以下のタスクがある。

1. 理論を強く必要としている(そして、現場の大部分ではいまだしっかりした理論に基づいた活動がされていない)ソフトウェア工学の領域を特定する。
2. すでにソフトウェアに適用されているが少数のプロジェクトでしか例がない理論の中から、広い応用への潜在的価値があるものを特定する。
3. タスク1で特定された領域から、まだ理論が存在しないものを特定する。
4. 特定の理論サポートを必要とする Semat 活動のコンポーネントを識別する。特に、カーネル言語(付録 4)にしっかりした理論を提供するのに必要な理論の種類を定義する。

この4つのゴールが1年の期間内に Semat プロジェクトの最初のマイルストーンとして到達可能であろう。次のステップでは、次のことが必要になる。

5. ソフトウェア工学理論の応用ロースター(Roster Of Applicable Software Engineering Theories: ROAST)を開発し、利用できる既存理論(上記の 2 点目で特定された)のリストと、そのそれぞれに、実プロジェクトでの詳細な適用ガイドラインを提供する。
6. カーネル言語に対する理論を開発する(上記 4 点目参照)
7. 開発すべき少数(3 つ以内、できれば 1 つ)の理論を選ぶ。その開発への条件を(可能であれば、それらの理論の基礎要素も)定める。

付録 3:　ユニバーサル[2](Universals)

カーネルを具体的で、焦点を絞った、小さいものに保つためには、ソフトウェア工学の真に普遍的な要素のみを特定する必要がある。そのような、すべてのソフトウェア工学活動に本質的な要素について、その存在は一般的に合意されているようだが、それらが何で、それらが本当に本質的かどうかどのように検証するのか(カーネルに含まれるかどうかの条件)については、大きな混乱がまだ残っているようだ。ユニバーサルタスクの役割は、これらの要素を特定することである。

A3.1 カーネルの特性

ここに、すべての成功するカーネルが備えていると私たちが考える特徴の候補を挙げる。

1. **簡潔(Concise)**。カーネルは真に本質的な少数の要素にフォーカスしていなければならない。
2. **スケーラブル(Scalable)**。カーネルのスコープは小さなプロジェクトから大きなシステム、さらに「システムのシステム」までをカバーしなければならない。
3. **拡張可能(Extensible)**。カーネルはプラクティス、パターン、詳細レベル(levels of detail)、ライフサイクルモデルを追加できる能力を提供しなければならない。特定ドメインへ適用とソフトウェア以外を含むプロジェクトへのテーラリングをサポートしなければならない。
4. **計測可能(Measurable)**。カーネルはすべての関連するソフトウェアプロセスとプロダクト成果物を定量的に評価する機構を提供しなければならない。
5. **形式的定義(Formally specified)**。カーネルは数学的に定義されていなければならない。この定義はカーネルとともに開発されなければならず、仮設的な将来の活動に延期させてはならない。この目標は「理論トラック」と協調して達成される。
6. **広範囲な実践カバレッジ(Broad practice coverage)**。カーネルは多種の異なるプラクティスを、産業界で有意なセグメントに有用性を認知されている範囲でサポートしなければならない。
7. **広範囲なライフサイクルカバレッジ(Broad lifecycle coverage)**。カーネルは様々なライフサイクルモデルを、産業界で有意なセグメントに有用性を認知されている範囲で収容しなければならない。
8. **広範囲な技術カバレッジ(Broad technology coverage)**。カーネルはソフトウェア技術の広いレンジ(プログラミング言語、仕様言語、グラフィックノーテーション、ソフトウェアツール)に、産業界で有意なセグメントに有用性を認知されている範囲で適用可能でければならない。

A3.2 ロール(Role)

カーネルは Seat initiative にとっての「試練」(crucible)である。なぜなら、カーネルがすべての他の

[2]原注: Ian Spence と共著

Semat 活動のソフトウェアエンジニアの実践的な仕事へ適用性を決定するからだ。カーネルは、すべてのプロジェクトで使える具体的なフレームワークを提供し、エンジニア自身の環境の中で必要なプラクティスを識別し適用するこができるようにする。

A3.3 判断基準(Criteria for inclusion)

採用されるべき要素を特徴づける、基本的なルールは以下のとおり。

- **普遍的(Universal)**: すべてのソフトウェア工学活動に潜在的に適用可能。
- **意味がある(Significant)**: ソフトウェアプロセスもしくはプロダクト(あるいは両方)の品質に、プラスにかつ認識可能に貢献できる能力を持つ。
- **課題に関連している(Relevant)**: 技術的なキャリアや信じる方法論の流派（camp)に関係なくすべてのソフトウェアエンジニアによってアプリケーションが利用できる。
- **定義されている(Defined)** 詳細に。
- **実行可能(Actionable)**: 言葉とコンセプトだけでなくプロジェクトが採用できる詳細なガイドライン。
- **検証可能(Assessable)**: 適用について定量的評価ができる。
- **包括的(Comprehensive)**: この基準がカーネルの要素のすべてに適用できる。さらに、ソフトウェア工学の本質を捉え、ソフトウェア工学チームの重要なプラクティス、パターン、メソッドを支援する地図を提供する。

A3.4 例と問い(Examples and questions)

Semat Web サイトにおける、カーネルに関する様々なブログやディスカッションで、いくつかのユニバーサルの候補が特定されてきた。ここで、いくつかの例を議論する。

何人かの貢献者は、どんなソフトウェア工学の活動においてもユニバーサルの1つとして、「動くプログラム」(working program)をあげている。これは何を意味するのだろう。定義されたテストを通過するプログラム？定義された要求を満たすプログラム？顧客のほんとうのニーズを満たすうプログラム？

「要求」(requirements)はどうだろう。要求はユニバーサルだろうかそれとも単に、顧客の意図を把握するために使われるプラクティスの1つだろうか？

このような問にこのワーキンググループが答えながら、最初のカーネルが形作られる。このカーネルは、われわれのもっともよく知られたプラクティスのいくつかを検討したり、ソフトウェア工学の定義に挑戦したりするのに使われる。

ほかの貢献者は、次のようなより多くのユニバーサル候補を挙げた。

- プロジェクト(Project)
- チーム(Team)
- ユーザエクスペリエンンス(User Experience)
- システム(System)
- 品質(Quality)
- 意図(Intent)

これらはすべてユニバーサルだろうか。これらはすべてソフトウェア工学にとって本質だろうか。これらは同じ概念レベルだろうか(「同じタイプのことがらだろうか」)。これらはお互いに関連しているか、そうだとしたらどのように。

カーネルタスクのゴールは、これらすべての質問に答えながらソフトウェア工学の具体的なモデルを作り、われわれソフトウェア工学のコミュニティをプロフェッショナルにするためのプラクティス、パターン、メソッドの定義、評価、そして改善の基礎を提供することである。

付録 4: カーネル言語(Kernel Language)

カーネル言語の考察はまだ初期段階なので、この付録はその言語に対する最終的な要求を定義していないが、どのようなニーズに対処するのか、というアイディアだけでもここに記述しよう。

カーネル言語の開発は、違った経験を持つ 2 種類貢献者を必要とするだろう。

- 1つ目は、カーネル言語がユーザニーズをカバーできるよう、メソッド要素(この付録で使われている用語で、メソッド、パターン、プラクティスのこと)の広範囲な経験を持つ。
- 2 つ目は、言語設計の経験を持つ。

このセクションの2つの付録は、上記の2つに対応してカーネル言語の利用法とその設計について書いている。

A4.1 カーネル言語の利用法

カーネル言語の役割は、プラクティス、パターン、それらが組み合わされたメソッドを記述することである。このセクションでは、「記述」(description)という言葉は、カーネル言語で表現されたメソッド要素(上で説明した意味で)の記述のことを指す。

カーネル言語の最も重要な想定ユーザは、適切なメソッド要素を探していて、選定したモデル要素を適用したいと考えているプロジェクトである。

カーネル言語とそれによる記述は、そのユーザが期待する価値を提供するために、以下のような特性を満足しなければならない。

- 言語は、われわれが関連するすべてのプラクティス、パターン、そしてその組み合わせとして**今日の(today's)**メソッドをカバーできる。
- それらを**組み合わせて(composing)**新しいメソッド要素を記述する、複数のやり方をサポートする。
- **拡張可能(extendible)**であり、今時点ではまだ発明されていないメソッド要素と(その中の1つ1つのプラクティスのような)詳細要素の記述を可能にする。
- 記述は**理解しやすい(easy to understand)**ものである。言語は開発者コミュニティのために設計され、プロセスエンジニアや研究者のためだけのものではない。
- 言語はメソッド要素同士の**比較**をサポートする(現在非常に難しいタスクの1つ)。
- 記述は Semat 活動(付録 A.3 参照)で特定された**ユニバーサル**の言葉を使って構築され、Semat の主要なゴールの1つ(プラクティス、パターン、メソッドの再発明を避けること)を助ける。
- 言語とその記述は、メソッド要素の適用を**模倣する(simulating)**ことを支援する。
- それらは、メソッド要素を現実のプロジェクトに**適用する(applying)**ことを支援する。
- それらは、**妥当性確認(validation)**の機構を提供し、あるメソッド要素(カーネル言語で記述されている)を適用したプロジェクトが本当にそれを適用していて、リップサービスだけではないことを検証することができる。この問題は、ときどき「ギャップ埋め」("closing the gap")(プロジェクトチームがやっている、と自分でいっていることと、本当にやっていることのギャップ)と呼ばれる。

最後の要求はチーム組織についての含みがある。すべてのチームは、プロジェクトの選択されたメソッド要素に関して、現実のパフォーマンスを評価するために適切な資源を提供するべきである。

A4.2 カーネル言語の設計

カーネル言語について事前の設計が存在するわけではないが、その設計について以下の要求が特定されてきた。

他のすべての厳密に定義された言語のように、カーネル言語は抽象的なシンタクス、妥当性ルール("static semantics")、そしてセマンティクスを持たなければならない。少なくとも1つの具体的なシンタクスが必要だが、テキスト形式とグラフィックス形式、のように複数でもよい。

カーネル言語は4つの主要な応用を支援しなければならない。

- ユニバーサルを**記述する(describing)**: プラクティスとパターンはビルディングブロックであり、メソッドをこれらから構築するための、組み合わせ機構である。
- 結果できた記述をプロジェクトに適用することを、**模倣(simulating)**できる。
- 現実に記述を**適用(applying)**し、「ギャップ埋め」(上記)ができる。

- メソッド要素のシミュレーションや適用を**評価(assessing)**できる。

仕事の進行状況を表現するのに、**状態(state)**の概念がカーネル言語で重要な働きを担うように思う。例えば、反復開発(iterative development)を含むようなプラクティスでは各イテレーションの開始と終了状態を記述する必要がある。

カーネル言語は**拡張可能(extensible)**でなければならない。特に、プラクティス、パターン、もしかしたら組み合わせテクニックも含めて、追加・変更できなければならない。

- プラクティスはメソッドの切り離された関心事(concern)である。例としては、「要求からテストまでの開発」(development from requirements to test)、「反復開発」(iterative development)、「コンポーネントベース開発」(component-based development)などである。
- すべてのプラクティスは、他のプラクティスから切り離して自己完結記述される。
- すべてのプラクティスは、明示的に継続的活動であると定義されない限り、明確な開始と終了を持つ。
- すべてのプラクティスは、ステークホルダへの定義された価値をもたらす。
- すべてのプラクティスは、評価可能(assessable)である。その記述は評価のための基準(criteria for its assessment)を含む。
- 可能であればいつでも、その評価基準は定量的な要素を含まなければならない。対応する記述は適したメトリクスを含まなければならない。
- プラクティスの組み合わせ機構は、マージと拡張を含む。

以下のルールがパターンに適用される。

- すべてのパターンは、(第 3 節の定義から)いくつかの違ったプラクティスに適用可能である。
- すべてのパターンは、他のパターンおよびそれが適用されるプラクティスから切り離して、自己完結記述される。

付録 5: 評価(Assessment)[3]

しっかりしたソフトウェア工学の基礎を確立するには、プラクティス、パターン、メソッド(これら 3 つはこの付録内では「メソッド要素」と呼ぶ)を評価する体系的な手段と、客観的な基準に基づいた支援ツールが必要である。他の工学分野と同様に、これらの基準は定量的でなければならない。健全な理論(付録 2:参照)に則っていなければならない。統計的に健全なデータを根拠にしなければならない。

この付録の 3 つのセクションで述べられている基礎的な問いは、以下である。

- 評価されたメソッド要素の目的は何か。

[3]原注: Watts Humphrey と共著

- それらの品質を判断する基準は何か。
- 測定プロセスはどのようなものであるべきか。

A5.1 評価目的(Assessment objectives)

カーネル、特に関連するメソッド要素(上記の意味で)、を特定したら、これらの要素がユーザの実際のタスクの役に立つかどうかを評価する客観的基準をユーザに示さなければならない。このことから、2つの疑問が起こる。「ユーザ」は誰か、そして、彼らの意図する「タスク」とは何か。

もっとも直接的に関連する**ユーザ**とは、

- ソフトウェア開発実践者(エンジニアとマネジャ)。彼らは実践的であり、自分で利用可能かつ使いやすいと考えるメソッド要素は何でも使うだろう。
- その顧客。彼らが一般的に関心をもつのは、品質のよいソフトウェアを速く、低価格で、予測可能な方法で取得することである。
- 学会。ソフトウェア工学の教育や研究をしている。

このすべてのユーザカテゴリが、機能性、有用性、コスト効率、サポート、ユーザビリティ、信頼性、効率性、拡張性、再利用性、といったすべての伝統的なプロセスとプロダクトの品質を大切にしている。

評価に関する**タスク**はソフトウェア工学のすべての重要なタスクを含む。技術系のタスク(分析から設計、実装、ドキュメンテーション、検証と妥当性確認(verification and validation)、配置(deployment)、オペレーション、その他)、および人間系のタスク(マネジメント、教育、サポート、コミュニケーション)の両方である。

A5.2 品質評価(Assessing quality)

評価方法は、すべてのプラクティスに適用可能であり、それらの重要な特性を考慮しなければならない。

- **機能性(Functionality)**: メソッドの記述は、意図した機能のカバレッジの測定を可能にするか。
- **ユーザビリティ(Usability)**: ユーザビリティはどうやって測定されるか。手法は、できればユーザビリティの研究所、あるいはユーザ調査を含む。このような調査は、さまざまなユーザの状況を測定すること、そして、統計的に有用で客観的なデータを複数の情報源から取得するために、精密に定義され再現可能にするべきである。
- **信頼性(Reliability)**: メソッド要素は常に意図した結果を作りだすか。プロジェクト変数は関係するか。そしてもしそうなら、どのような変数があり、その見込みと影響はどうか。
- **効率性(Efficiency)**: この生産性の尺度は、そのメソッド要素を適用するのにかかった一人もしくはそれ以上のソフトウェアエンジニアの使用した時間のデータ、およびプロダクトが作り出した仕事量を必

要とする。

- **拡張性(Extensibility):** これは、実装が将来の成長を考慮にいれるというシステムの設計原則である。これはシステムを拡張する能力および拡張を実装するのに必要な労力レベルの、システミックな尺度である。拡張は新しい機能の追加もしくは既存の機能の修正によって行われる。メソッド要素が、既存のシステム機能への影響は最小限に変化を提供する、という中心テーマをサポートするか。
- **再利用性(Reusability):** 定量的指標は4つのタイプからなる。(1)再利用のコスト対メリットモデル。経済的コスト対メリット分析、および品質と生産性のペイオフを含む。(2)再利用量のメトリクス。ライフサイクルオブジェクトの再利用率トラッキングによって、再利用率改善活動を評価、モニターできる。(3)再利用性メトリクス。ある成果物が再利用できる可能性を示す。(4)再利用ライブラリメトリクス。再利用リポジトリが利用されたことを管理、追跡するのに使用する。

A5.3 評価プロセス(Assessment process)

評価プロセスの定義には、3 つの側面がある。メソッド要素を評価する対象プロジェクトをどのように選択するか。どのような特性を測定するか。プロセスそのものをどう編成するか。以下ではこれらの課題を順に検討する。

A5.3.1 サンプルプロジェクトの選択

プラクティスと他のメソッド要素を評価するには、産業界での実際のプロジェクト、もしくは、実験的環境で管理されたテスト(例えば生徒を使う)、のどちらかを選択することができる。どちらの手法も完全に満足だとはいえない。

産業界のプロジェクトは 2 つの利点がある。規模(学術の現場では得るのは難しい)とリアリズム(実際の利用を意図したものだから)である。しかし、2つの大きな制限がある。1つ目は、通常それが再現性を持っていないこと。産業界プロジェクトのサイズと経済的な影響は、一般的には一回限りの活動であり、統計的価値に疑問がある孤立したデータポイントとなる。この問題は、1つの一貫したコンテクストにある多くのプロジェクトから評価データを採ることで軽減される。例えば1つの会社からすべて同一領域のプロジェクトを選ぶなど。もう1つの制限は、産業プロジェクトでは多くのパラメータがそのコンテクストにおいてセットされ、評価手順の管理下にないことである。

- 新プロダクト開発か既存プロダクトの修正か。
- 独立したシステムか大きなシステムの一部か。
- 企業の内部利用か販売目的か。
- カスタム開発かマスプロダクトか。
- アプリケーション領域と評価するメソッド要素の両方に関するチームのスキル、経験、教育。

- コストモデル：固定金額、コストプラス、あるいは他。(fixed-price, cost plus or other)
- 評価手順それ自身の影響(ハイゼンベルグ効果)

管理された実験は産業的なコンテクストにおいて投資することが難しいため、そのような実験は通常、アカデミアにおいて学生を使って行われる。この利点は、実験者が上記に挙げたパラメータをよりきめ細かに制御することができ、そしてもっとも重要なことは、ある程度の再現性を得ることができることだ。不利な点は、一般的に実験は、評価されたメソッド要素の小規模利用にしか当てはまらず、現実的な状況への一般化(すなわちスケーラビリティ)への課題を残す。また、生徒は必ずしも職業としてのソフトウェアエンジニアの代表ではないことも挙げられる。

おのおのの手法 – 現実プロジェクトでの評価とアカデミック環境での評価 – はそれぞれ利点と制限があり、十分に説得性のある評価には両方が必要である。すなわち、研究者による制御された実験によって基礎的な洞察を得、さらに産業界のプロジェクトで妥当性を試すことで、その洞察を産業的利用にスケールアップできることを確認する必要がある。

A5.3.2 何を計測するか

評価手順は、何を計測するかを精密に定義する必要がある。重要なプロジェクトと開発者に関する変数、アクティビティ、1つ1つのアクティビティに費やされる時間、発見された欠陥、生産されたワークプロダクト。

A5.3.3 評価システムの構築

効果的で便利な評価システムの構築には、以下の2つの主要なコミュニティの支援を得ることが必要である。

- ソフトウェアエンジニアのコミュニティおよび実践者に対し、ソフトウェア工学のメソッド要素と成果(プラクティス、パターン、メソッド、ツール)を客観的、定量的に評価することの価値を説得すること。
- 学術的な研究の重要性から(A5.3.1)学会のコミュニティに対して、このタスクの価値を説得し、アカデミックが彼らの学生に厳密なデータ収集の手法を教えることを支援する。さらに、得られた成果(「実験的コンピュータ科学」(experimental computer science)と呼ばれる一般学術領域に属する)が、学問への貢献度の基準によって、適切な学術的認識を特にその公開において得ること。 (例として、自然科学では、別の研究者が過去に公開された結果に対して確認したり否定したりする、再現実験(reproducibility experiments)を、さらなる公開価値あるもの、さらには潜在的に名誉のあるものとして受け入れる、という伝統がある。コンピュータ科学のほとんどの領域では、この伝統が欠如している。