

要求工学の理論化に向けて

佐伯 元司^{1,a)}

概要：本稿では、要求工学分野の知識を記述し、蓄積し、活用するための理論化の手法を例示する。

Toward Theorizing Requirements Engineering

1. はじめに

ソフトウェア工学は、従来の自然科学と違って、ソフトウェアという人造物を相手にしているため、原理、法則などがなく、それゆえ問題に対する万能的な解を見つけるのが難しい学問であると言われている [1]。しかしながら、これまで多くのソフトウェア工学技術が考案され、ある状況下で現場でも効果を発揮しているものもある。これらの技術を効果的に適用するための経験や知見を知識化し、蓄積していくことが、現在のソフトウェア開発の問題を解決するうえで重要である。このようなソフトウェア工学分野の知識を収集・蓄積し、理論として体系化し、開発現場に活用しようとする動きがある。Jacobson らの SEMAT [2], ICSE の併設ワークショップ GTSE (General Theory of Software Engineering) での成果などである。理論を適用することにより、これから開始しようとする開発プロジェクトや進行中のプロジェクトで、将来起こりうる現象を予測し、適切なソフトウェア工学技術を効果的に適用することが可能になってくる。本稿では、Sjøberg らが提案した手法 [3] を用い、要求工学の中の要求獲得プロセスに関する知見 [4] の例を記述する。

2. Sjøberg らの手法

図 1 は、理論と現実世界の関係を書いたよく知られているもので、現実世界 (Real world) で現象 Situation が起こったときに現象 Result が観測された場合、Situation によって特徴づけられた構成要素 Assumption Construct が、Result によって特徴づけられる構成要素 Conclusion に影響を及

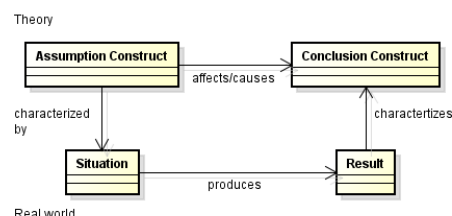


図 1 理論と現実世界

ぼしたり、Conclusion を引き起こしたりする。図の上部の理論部分 (Theory) を記述するためには、Construct と、その間の影響を及ぼす関係や因果関係である affects/causes 関係を定義できればよいことになる。Construct はソフトウェア工学特有の概念要素である。

Sjøberg らは、上記の Construct と影響や因果関係 (Proposition と呼んでいる) に加えて、なぜその Proposition が成り立つのかという説明 (Explanation) とその理論の適用範囲 (Scope) を導入している。Construct はさらに、Actor, Activity, Software System, Technology の 4 つに分類されている。これは、Actor が Technology を用いて、Activity を行い、Software System を開発した（もしくは Software System に対し Activity を行った）ということの意味する。

3. 適用例題

本稿では、要求工学の中の要求獲得プロセスに関する知見を Sjøberg らの手法を用いて、記述した。この例は、インタビューによるレストランのサービス・注文管理システムの要求獲得で、獲得した要求を 5 つに分類し、それらが獲得プロセスのどの段階で獲得されたかを分析した例である。この例に関連する Construct を図 2 にクラス図として示す。開発対象はレストランサービス・注文管理シス

¹ 東京工業大学
Tokyo Institute of Technology, Ookayama, Meguro, Tokyo
152-8552, Japan

^{a)} saeki@se.cs.titech.ac.jp

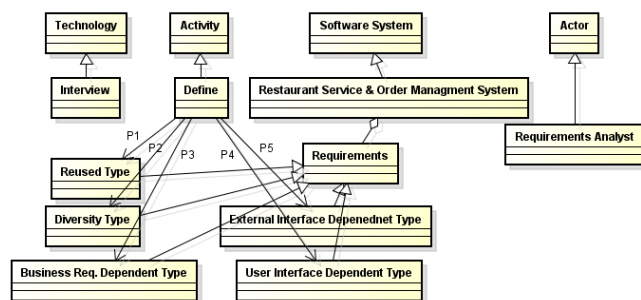


図 2 要求獲得プロセス例の構成要素

テム (Restaurant Service & Management System) であり、これが Software System のサブクラスとなっている。インタビュー手法 (Interview) を使って、このシステムの要求 (Requirements) を定義する (Define) ことがこの理論の対象である。Define は Activity, Interview は Technology のサブクラスになっている。Requirements は、5 つに分類され、Reused Type はあとで変更されて再利用される要求、Diversity Type はソフトウェアの構成要素のバリエーションに関する要求、Business Req. Dependent Type はビジネス戦略に関する要求、User Interface Dependent Type, External Interface Dependent Type は、各々ユーザインターフェース、外部システムとのインターフェースに関する要求である。図中の P1, ..., P5 は、Construct 間の影響を及ぼす関係 (affects/causes) で、Proposition である。例えば、P1 は Reused Type の要求 (Requirements) とそれを定義する (Define) 行為の間の関係を表し、「Reused Type の要求は早期段階で定義される」ことがその内容である。これは、定義する (Define) という Construct が Reused Type の要求 (の獲得時期) に影響を与えることを主張している。P1, ..., P5 が [4] で得られた知見である。Proposition, Explanation, Scope を表 1 に示す。例えば、Explanation E1 「プロセス中で変更されて再利用される要求は、他の要素への要求とは独立であるため」は、P1 の理由を説明している。これは、他の要素への要求とは関係なく獲得できるため、他の要因を考慮することなく、早期に定義でき、後段で変更されて再利用できるためであるとしている [4]。以下、E_i (1 ≤ i ≤ 5) は P_i の理由を説明している。

4. 議論

図 1 で示した上部の Theory 部分が前節で記述したものであり、下部の Real world 部分が [4] で実験的分析で示されている部分である。現在のソフトウェア工学分野の論文は、実験・経験による実証や評価を綿密に行っているものが主流になっている。そこで得られた知見を、ソフトウェア開発の現場に適用できるように理論化し、その論文の成果を理論の証拠として対応付け、蓄積していく活動が重要である。要求分析、特に要求獲得段階は、ソフトウェア開発中で人間の創造的な能力がもっとも発揮される段階

表 1 理論の記述

Proposition	
P1	Reused Type の Requirements は早期段階で定義 (Define) される。
P2	Diversity Type の Requirements はプロセス中で継続的に定義される。
P3	Business Req. Dependent Type の Requirements は早期段階で定義されたり、プロセス中で継続的に定義されたりする。
P4	User Interface Dependent Type の要求は、プロセス中で継続的に定義される。
P5	External Interface Dependent Type の要求は、後期段階で定義される。
Explanation	
E1	プロセス中で変更されて再利用される要求は、他の要素への要求とは独立であるため。
E2	分析者がシステムのバリエーションに関する知識を十分に持っていないため。
E3	早期段階で定義されるのは最初に顧客の動機づけを明確にする必要があり、継続的に定義されるのは競合システムを意識することになっていくためである
E4	プロトタイプングやテストによる確認をやりつつ、定義されていくため。
E5	使用する外部システムやその仕様が明らかにしないと (例えば、ベンダーからの情報を収集するなど)、要求が定義できないため
Scope	
要求の定義 (Define)、インタビュー (Interview) を使用、レストランサービス・注文管理システム (Restaurant Service and Order Management System)	

であり、それゆえ、人間が得た知見は暗黙化されていることが多い。これらを明示し、理論化しておくことは、他の段階よりも必要性が高いと考えている。

理論の記述には、Sjøberg らの手法を用いたが、本質的部分は Construct 間の影響・因果関係 (Proposition) を基本的には 2 項関係で記述するのみにになっている。Construct と Proposition, Proposition 同士について関係をつけることも可能である。しかし、関係をつけるにあたっては、何が原因で何が結果かという因果関係を明確につきとめる必要がある。今回の例では、要求の定義 (Define) 行為が原因で、要求のタイプごとの定義される時期が結果であるという解釈をとったが、Interview 手法を使用したことが主たる原因であるとみなすこともできる。今後は、関係の抽出や表現手法の洗練化が課題となってくるであろう。

参考文献

- [1] Brooks, F. : *No Silver Bullet ? Essence and Accidents of Software Engineering*, IEEE Computer, 20(4), pp.10-19, 1987.
- [2] SEMAT: Software Engineering Method and Theory. 入手先 (<http://semat.org/>)(2016.12.4).
- [3] Sjøberg, I. et. al. : *Building Theories in Software Engineering*, Guide to Advanced Empirical Software Engineering, Springer, pp.312-336, 2008.
- [4] Nakatani, T. et. al. : *A Case Study: Requirements Elicitation Processes throughout a Project*, RE2008, pp.241-246, 2008.