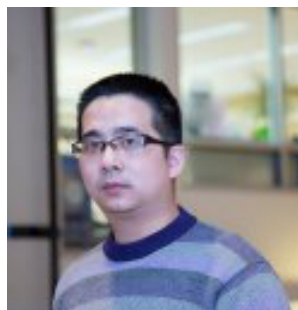




How Practitioners Perceive Coding Proficiency



Xin Xia, Zhiyuan Wan, Pavneet Singh Kochhar, David Lo

Proceeding [ICSE '19](#) Proceedings of the 41st International Conference on Software Engineering Pages 924-935

発表者 吉富 楓雅

開発者に必要なスキル

ハードスキル	ソフトスキル
体系だった知識	ヒューマンスキル
理論・手法・ツール	自己および対人関係に関するスキル
定量化可能	定量化が困難

コードを作成するために、
ハードスキルとソフトスキルに優れた開発者が求められている

ハードスキル

- ハードスキルはコーディングの習熟度に貢献する可能性が高い
 - * コーディングの習熟度を向上させるためのツール[1][2]
- 絶えず変化していくIT業界の中で…
 - * どのようなハードスキルが必要か？
 - * どのハードスキルに投資すれば良いのか？

[1] J. Bishop, R. N. Horspool, T. Xie, N. Tillmann, and J. de Halleux, "Code hunt: Experience with coding contests at scale," in 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2, 2015, pp. 398–407.
[2] N. Tillmann, J. de Halleux, T. Xie, and J. Bishop, "Pex4fun: A web-based environment for educational gaming via automated test generation," in 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013, 2013, pp. 730–733.

目的とアプローチ

- 目的

- * どのようなハードスキルがコーディング能力に貢献するか調査

- アプローチ

- * 15人に対するインタビュー，および340人の開発者にEメールによるアンケート調査

ハードスキルの分類

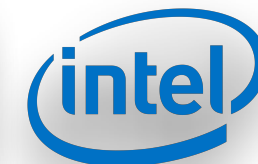
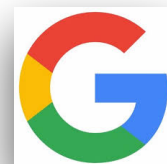
1. 一般的なコーディングスキル：(例)効率的なコードの書き方，文書化されたコード
2. プログラミング言語とインフラ：(例)複数言語の習得
3. リファクタリングと再利用：(例)再利用可能なコードの抽出
4. 要求工学：(例)要件からのモデル抽出，要件に応じた正しい機能実装
5. ソフトウェア設計：(例)モジュール設計に従った機能実装
6. 理解と学習：(例)短期間での既知コードの理解，新しい言語の早期学習
7. 環境との相互作用：(例)複数のIDE・OSでの作業，バージョン管理システムの操作
8. バグの予防と修正：(例)迅速かつ正確なバグ修正
9. 見積もり：(例)コード実行に必要な時間コストの見積もり

調査対象者

- 中国のIT会社 3 社から最終的に15人がインタビュー対象
 - * ソフトウェア開発の習熟度
 - * コーディング能力の重要性と向上のための方法
- 15人に対するインタビュー結果をEメールでアンケート調査
 - * 各スキルに対してLikert Scoreによる5段階評価(全く重要でない～非常に重要)
 - * 340人から回答を取得

ID	Experience	Role	Company
P1	12 years	Chief Architect	Hengtian
P2	10 years	Project Manager	Hengtian
P3	11 years	Senior Developer	Hengtian
P4	8 years	Senior Developer	Hengtian
P5	10 years	Senior Tester	Hengtian
P6	2 years	Junior Developer	Hengtian
P7	1 years	Junior Developer	Hengtian
P8	3 years	Junior Tester	Hengtian
P9	9 years	Project Manager	IGS
P10	10 years	Senior Developer	IGS
P11	13 years	Project Manager	SS
P12	8 years	Team Leader	SS
P13	9 years	Senior Developer	SS
P14	2 years	Junior Developer	SS
P15	2 years	Junior Tester	SS

15人のインタビュー対象者[3]



リサーチクエスチョン

1. 開発者はコーディング技能スキルをどのように認識しているか？
2. 開発者が重要と考えるコーディング能力のスキルは何か？
3. 開発者が特定のコーディング能力のスキルを重要と認識する根拠は何か？

RQ1:開発者はコーディング技能スキルを どのように認識しているか？

Likert Scoreの平均値

C1. General Coding Skills		
S1	Write code efficiently (i.e., clear coding task in a short amount of time)	4.03
S2	Write efficient code, e.g., the code can run very fast, use less memory	4.12
S3	Write well-documented code	4.07
S4	Write parallel programs that leverage multiple threads and processes	3.56
S5	Write code that embeds well with code written by others, e.g., write code for a large project team which has many developers and require much collaboration	4.37
C2. Programming Language and Infrastructure		
S6	Master multiple program languages	3.61
S7	Master legacy programming languages (e.g., Cobol)	2.36
S8	Master popular programming languages (e.g., Java)	3.47
S9	Master big data infrastructure (e.g., Hadoop, elastic search)	3.15
C3. Refactoring and Reuse		
S10	Recognize and extract reusable code from a larger code base	4.16
S11	Package, document and distribute a software library for others to reuse	4.13
S12	Able to reuse code created internally rather than reinventing the wheel	4.19
S13	Reuse suitable third party libraries rather than reinventing the wheel	4.26
S14	Refactor code by identifying and eliminating code and architecture smells	4.28

RQ1:開発者はコーディング技能スキルを どのように認識しているか？

Likert Scoreの平均値

C4. Requirement Engineering	
S15 Extract an abstraction or a model from a requirement (e.g., in UML format)	3.53
S16 Implement a functionality correctly according to the requirement	4.31
S17 Recognize mismatches between requirement and implementation	4.29
C5. Software Design	
S18 Break down a complex coding task into smaller tasks that can be implemented separately	4.45
S19 Implement functionality following a modular design	4.34
S20 Implement functionality following suitable design patterns (e.g., singleton, factory)	3.71
S21 Implement functionality avoiding design anti-patterns (e.g., god class, brain class, feature envy)	3.82
C6. Understanding and Learning	
S22 Understand existing code in a short period of time	4.04
S23 Understand trade-offs between different system architectures	4.02
S24 Read and understand most books/articles related to the programming languages the developer mastered	3.88
S25 Acquire new domain-specific knowledge or learn new programming languages fast	4.03
S26 Learn the usage of new tools fast	4.04

RQ1:開発者はコーディング技能スキルを どのように認識しているか？

Likert Scoreの平均値

C7. Interacting with Environments	
S27 Work with multiple IDEs (such as Eclipse and Visual Studio)	3.05
S28 Work with multiple OSes (such as Linux, Windows)	3.38
S29 Modify the programming environment to tailor it to the developers' personal styles	3.64
S30 Work with a version control system (e.g., SVN, Git)	4.45
S31 Work with a bug tracking system (e.g., Jira, Bugzilla)	3.82
S32 Work with a code review system (e.g., Gerrit)	3.64
C8. Bug Prevention and Fixing	
S33 Locate and fix bugs fast and accurately	4.33
S34 Write good unit test cases to detect potential bugs	4.18
S35 Write good integration test cases to detect potential bugs	4.02
S36 Write good system test cases to detect potential bugs	3.95
C9. Estimation	
S37 Estimate well the effort needed to implement a new functionality	3.83
S38 Estimate the space and time cost of executing a piece of code	3.92

RQ1:開発者はコーディング技能スキルを どのように認識しているか？

- テスト担当者は、コーディング技能のスキルを他の開発者やプロジェクトマネージャーよりも重要だと考えている
- プロフェッショナルな開発者は、オープンソースの開発者よりもコーディング技能のスキルを重視している
- 学歴レベルが高い開発者は、コーディング技能スキルをより重視している

RQ2:開発者が重要と考える コーディング能力のスキルは何か？

- **Scott-Knott Effect Size Difference (ESD) 検定**を適用
 - * 38のスキルそれぞれで算出されたLikert Scoreの平均値を統計的にランク分割
 - ▶ 9段階に分割
 - ▶ ESD検定では, "わからない"と回答したものは除外

RQ2:開発者が重要と考える コーディング能力のスキルは何か？

重要度 高



Group	Skill
1	S18: break down a complex coding task into smaller tasks S30: work with a version control system S19: implement functionality following a modular design S5: write code that embeds well with code written by others S33: locate and fix bugs fast and accurately S17: recognize mismatches between requirement and implementation S16: implement a functionality correctly according to the requirement S14: refactor code by identifying and eliminating code and architecture smells
2	S13: reuse suitable third party libraries S12: reuse code created internally S34: write good unit test cases to detect potential bugs S10: recognize and extract reusable code from a larger code base S11: package, document and distribute a software library
3	S22, S26, S3, S23, S35, S25, S1, S36
4	S38, S24, S31, S37, S21
5	S20, S32, S29, S6, S4
6	S15, S8, S28
7	S9: master big data infrastructure
8	S27: work with multiple IDEs
9	S7: master legacy programming languages

RQ3:開発者が特定のコーディング能力の スキルを重要と認識する根拠は何か？

最高ランクのスキル



- 複雑なコーディングタスクを小さなタスクに分割(S18)
 - * 開発者が複雑な問題を解決するのに役立つ
 - * 開発者が再利用可能で保守可能なコードを作成できる
 - * チームメンバー間でタスクを効果的に分配することをサポートする
- 特に単体テストを実施する際に必要な作業

RQ3:開発者が特定のコーディング能力の スキルを重要と認識する根拠は何か？

最低ランクのスキル



- レガシー言語を習得する(S7)
 - * 全ての開発者が必要なスキルではないと回答
 - * コーディング業務の大部分でレガシー言語は必要とされない
 - * 新しい技術の習得に時間を努力を割くべきで、レガシー言語の習得は新しい技術のスキル向上に役立たない

RQ3:開発者が特定のコーディング能力の スキルを重要と認識する根拠は何か？

意見が分かれたスキル



- **文書化されたコードを書く (S3)**

- * 賛成者

- ▶ 再利用可能, 新人の参加促進, プログラム進化の支援

- * 反対者

- ▶ コードが明確に構造化されていて開発者が把握できているなら必要なスキルではない

まとめ

- 15人にインタビュー， 340人にEメールでアンケート調査
 - * コーディングの習熟度に関する調査を実施
- 9カテゴリ38のコーディング能力のスキルに分類
 - * それぞれのスキルの重要度を評価
 - * 評価の論理的根拠を示した

所感

-  • ソフトウェア工学で行われてきた研究に対して、実際の開発者がどう考えているのかがわかった
 - * 多くの人から調査しているので、信頼度が高そう
 - ▶ たまにスキルに対する考え方の違いが見られて面白かった
-  • 日本がすごく遅れている…
 - * レガシー言語は学ばなくて良いと他国は考えているよ…

付録

Likert Scale

- アンケートなどで使用される心理検査的回答尺度の1種
 - * 提示された文に回答者がその程度合意できるかを回答
 - * 肯定的反応や否定的反応を測る
- 5段階, 7段階, 9段階の尺度が存在
- 単独の項目の回答は順序尺度データとして扱い, ノンパラメトリックな検定で分析する
 - * 5段階程度では, 隣接する選択肢の間隔が等間隔であると感じられるかが不明であるため

あなたは組織の運営に対してどのくらい満足していますか？

☐	☐	☐	☐	☐
全く満足していない	あまり満足していない	どちらとも言えない	やや満足している	非常に満足している

Scott-Knott Effect Size

Difference (ESD) 検定[4]

- TantithamthavornらがScott-Knott 検定[5]を改良
 - * Scott-Knott testは正規分布に従うことが前提
 - * 正規分布に従わない場合は使用できない
- 入力データセットの非正規分布を修正するESD検定を提案
 - * 階層的クラスタリングを利用して、一連の処理手順を統計的に異なるグループに分割

[4] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Model," IEEE Transactions on Software Engineering (TSE), 2016.

[5] A. J. Scott and M. Knott, "A Cluster Analysis Method for Grouping Means in the Analysis of Variance," Biometrics, vol. 30, no. 3, pp. 507-512, 1974.