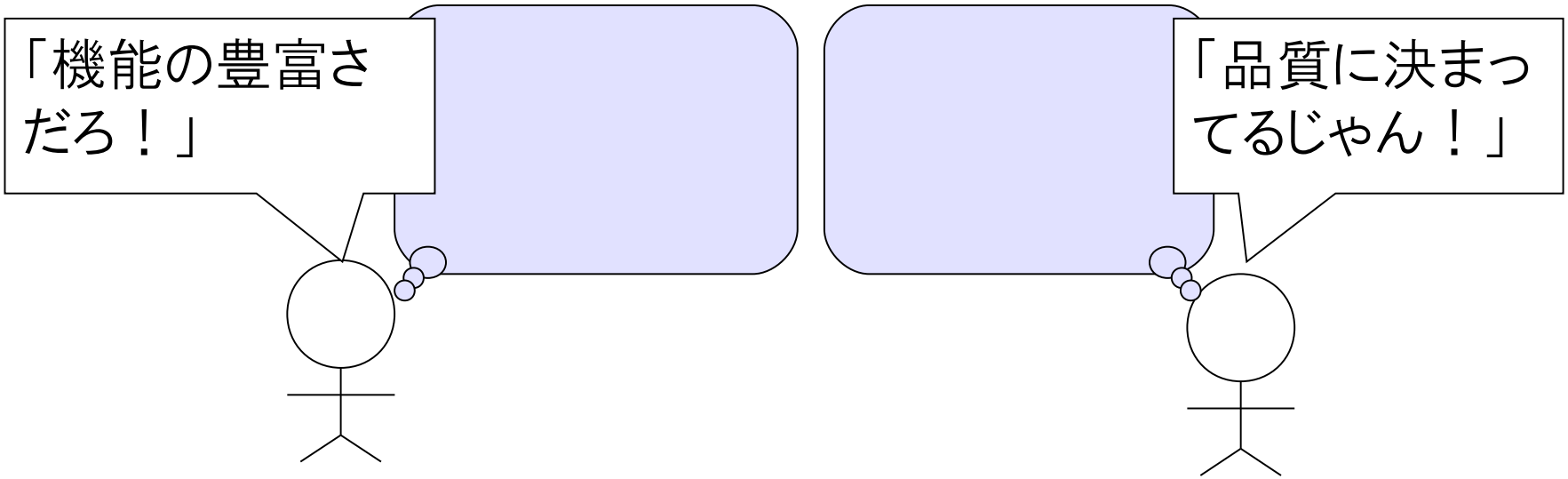
製品、ソフトウエア、プロジェクトの前提と品質の関連付け

森崎　修司
静岡大学 情報学部

ismoris@ipc.shizuoka.ac.jp
tweets: @smorisaki

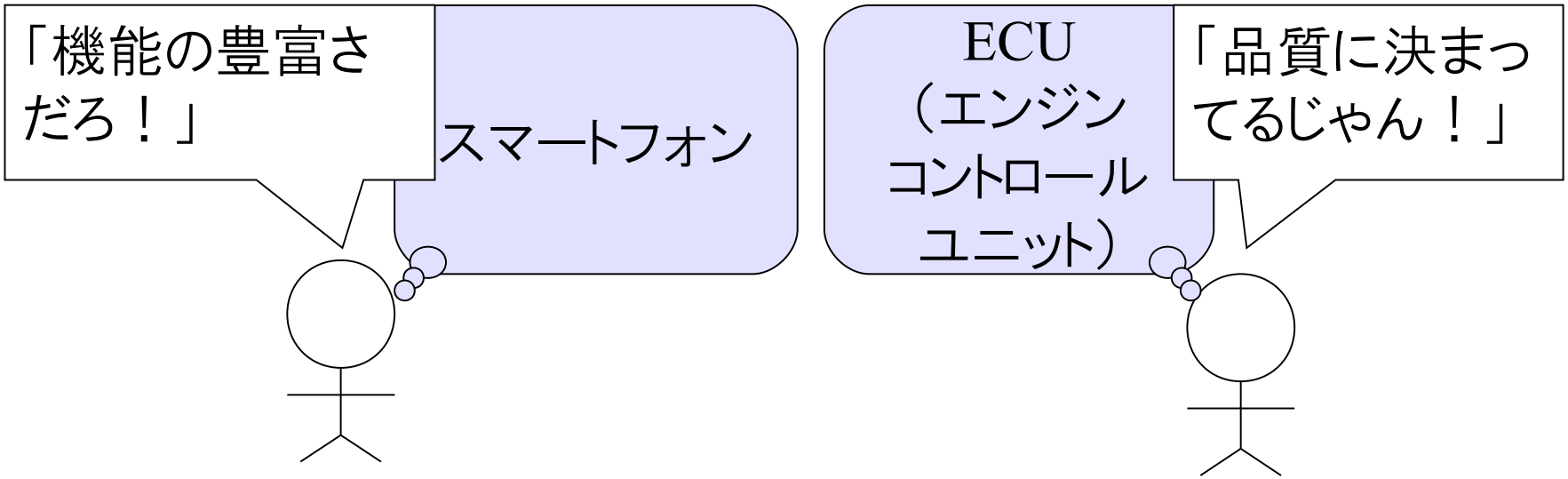
質問: ソフトウェアに求められるもの

- 前提や文脈なしの議論で・・・



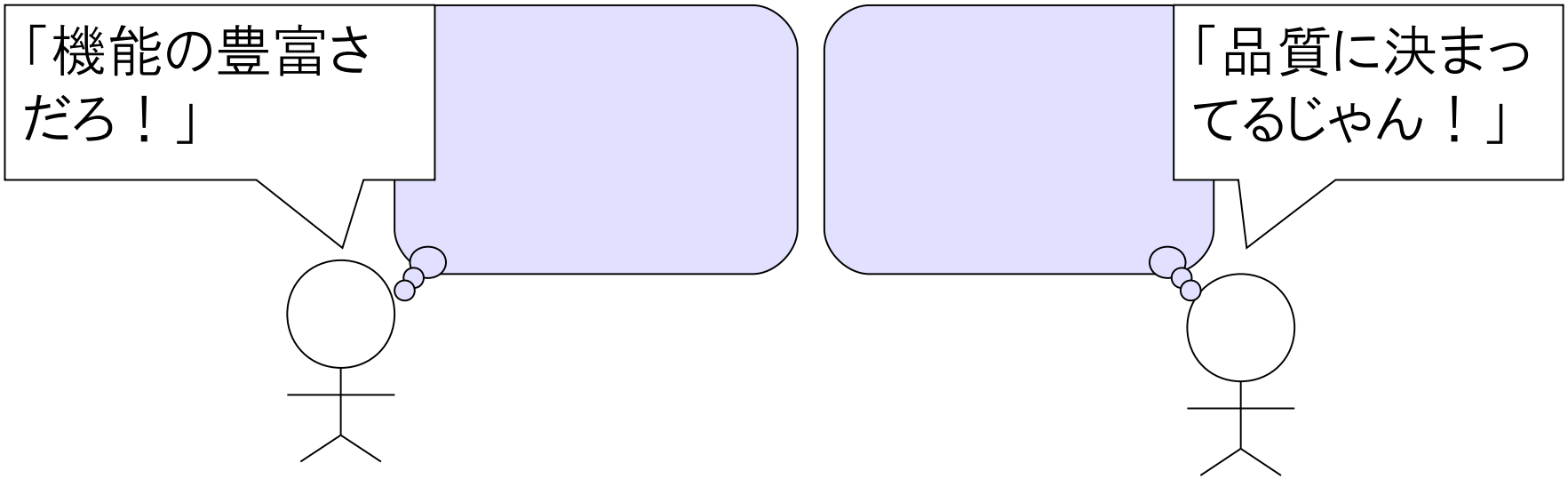
質問: ソフトウェアに求められるもの

- 現実にはこんなことも・・・



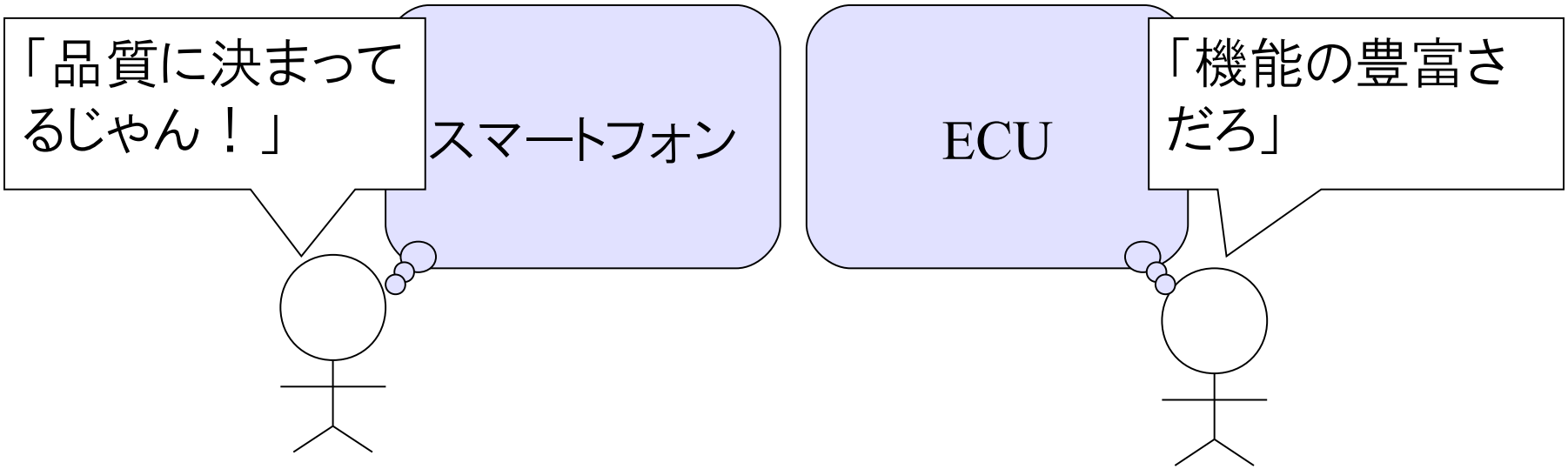
質問: ソフトウェアに求められるもの

- お互いのコンテキストに気づかずに、その議論を持ち帰ると・・・



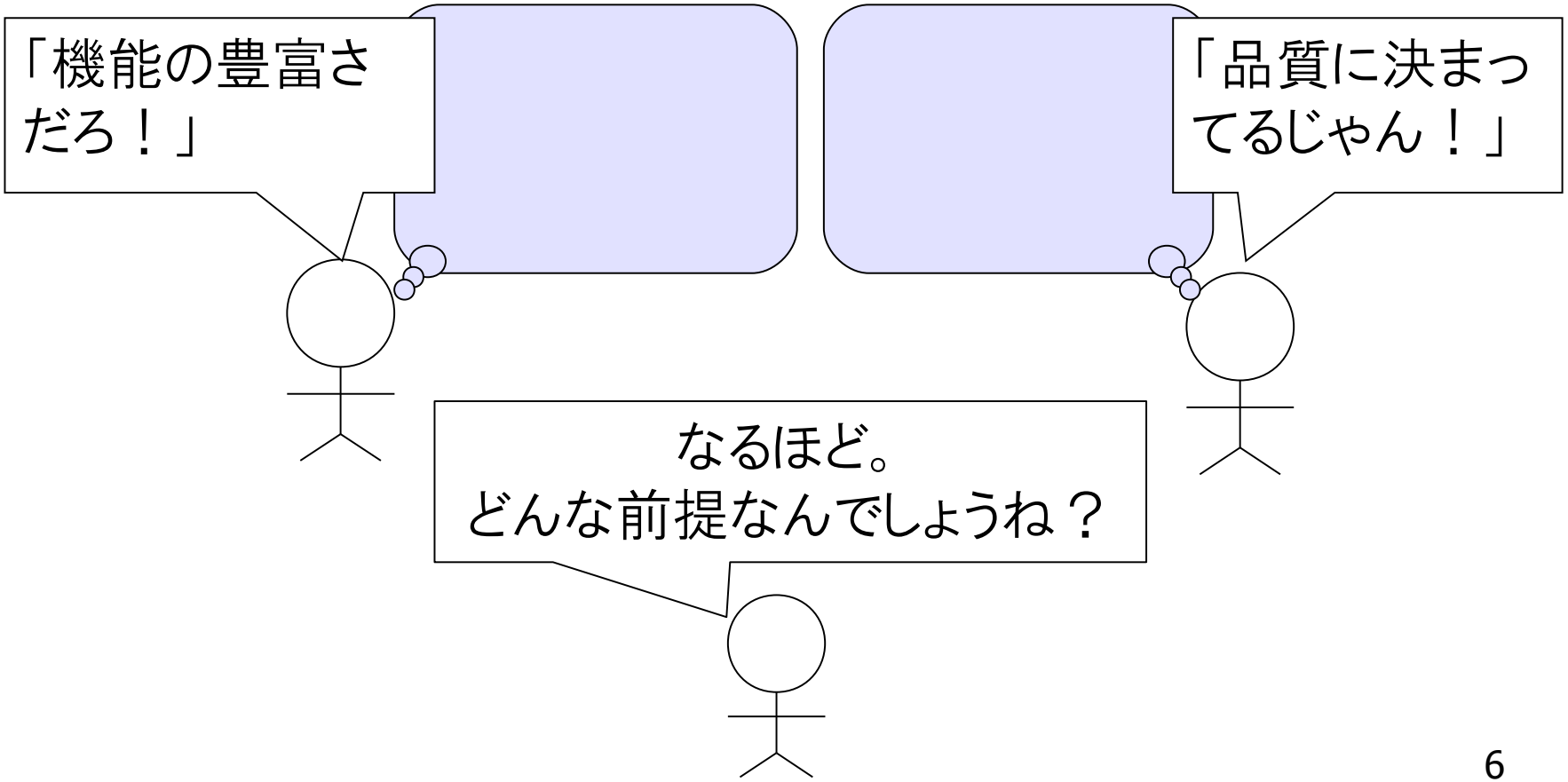
現実には起こりませんが・・・

- 間違えて現場に持ち帰る可能性もゼロではありません。



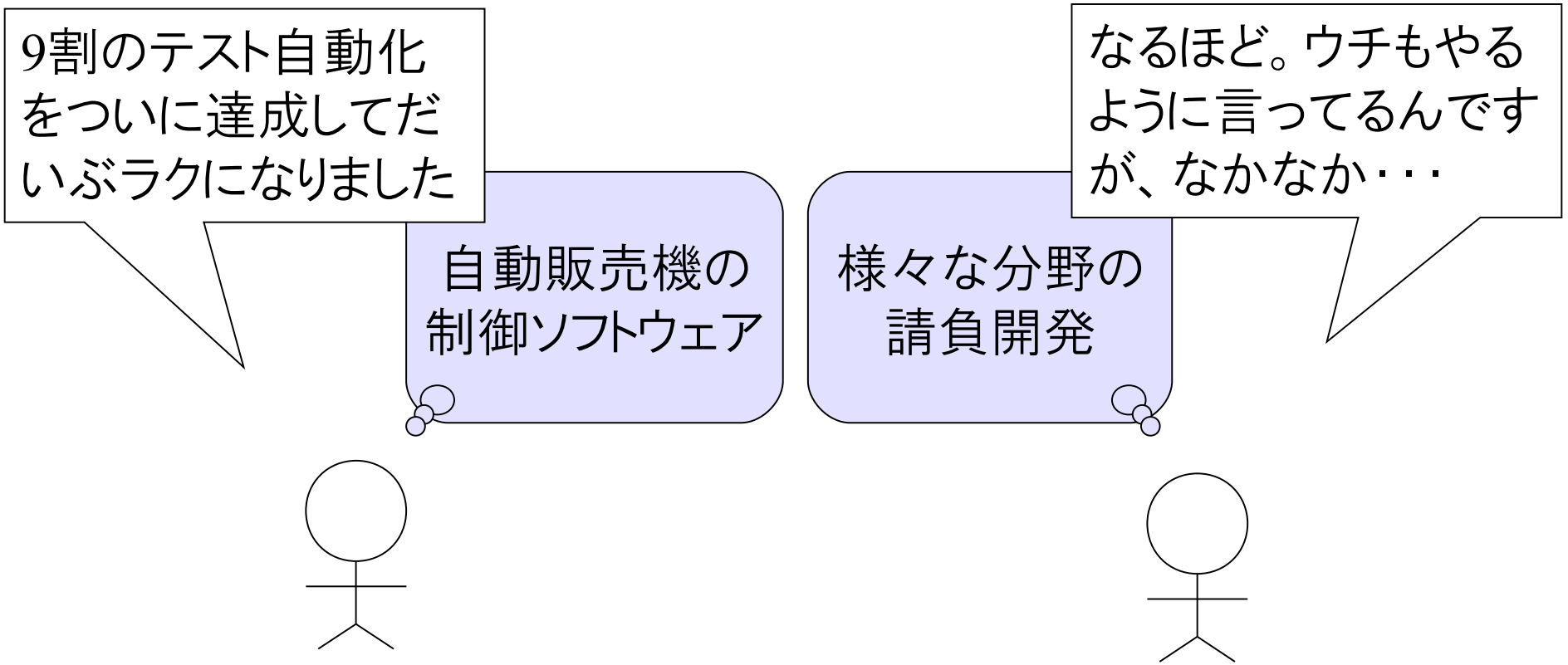
互いの前提を認識しましょう！

- どういう前提･制約で意見を述べているのか、さわやかに聞いてみましょう。



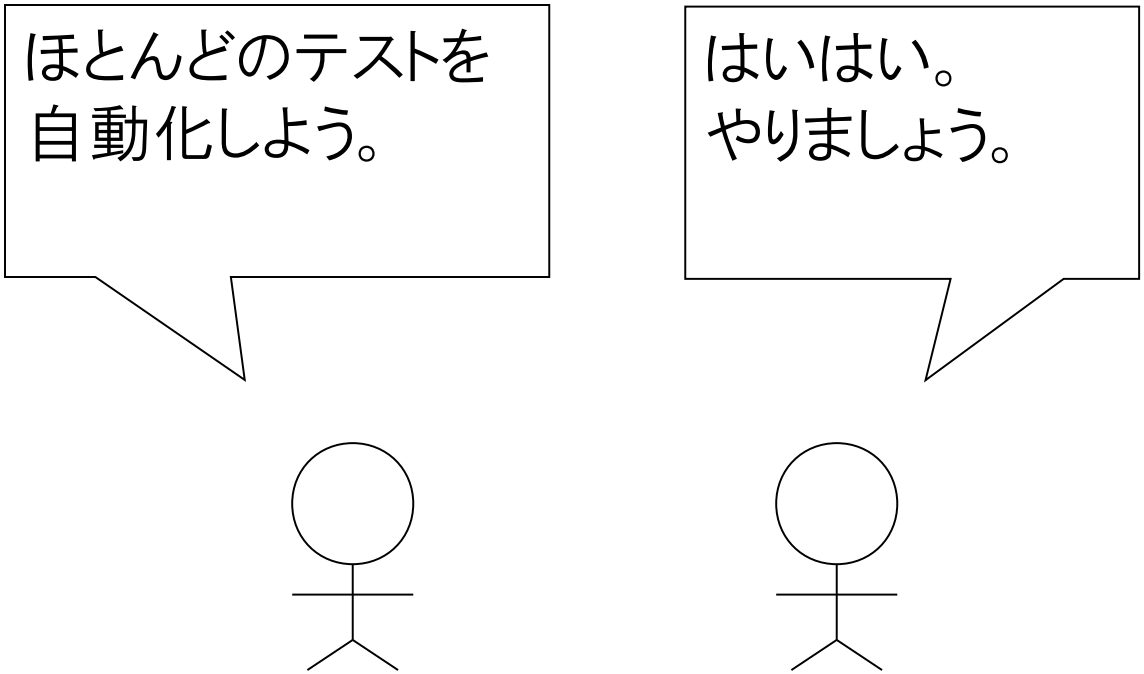
もう少しありそうな話を・・・

- 懇談会、委員会、連合会、連絡会議等で・・・・



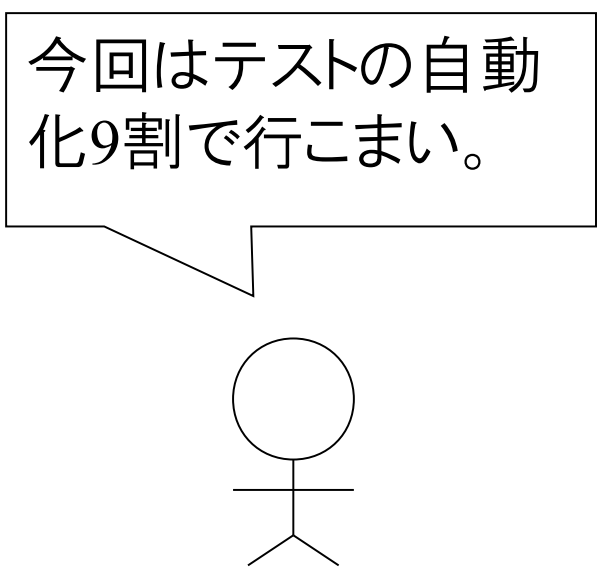
もう少しありそうな話を・・・

- 会社に戻り・・・

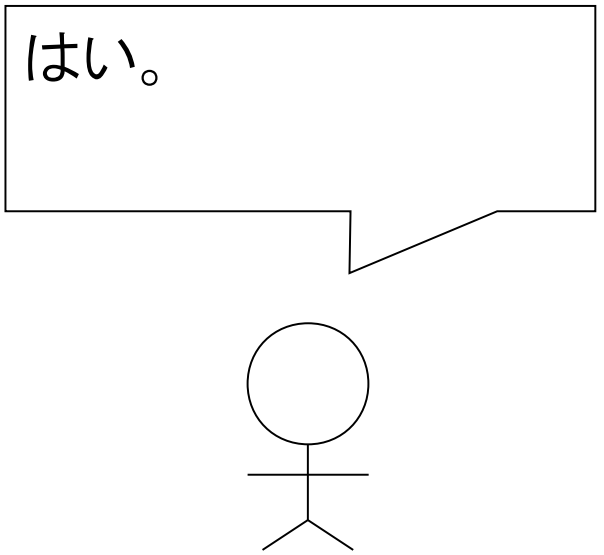


もう少しありそうな話を・・・

- 次のプロジェクトで・・・



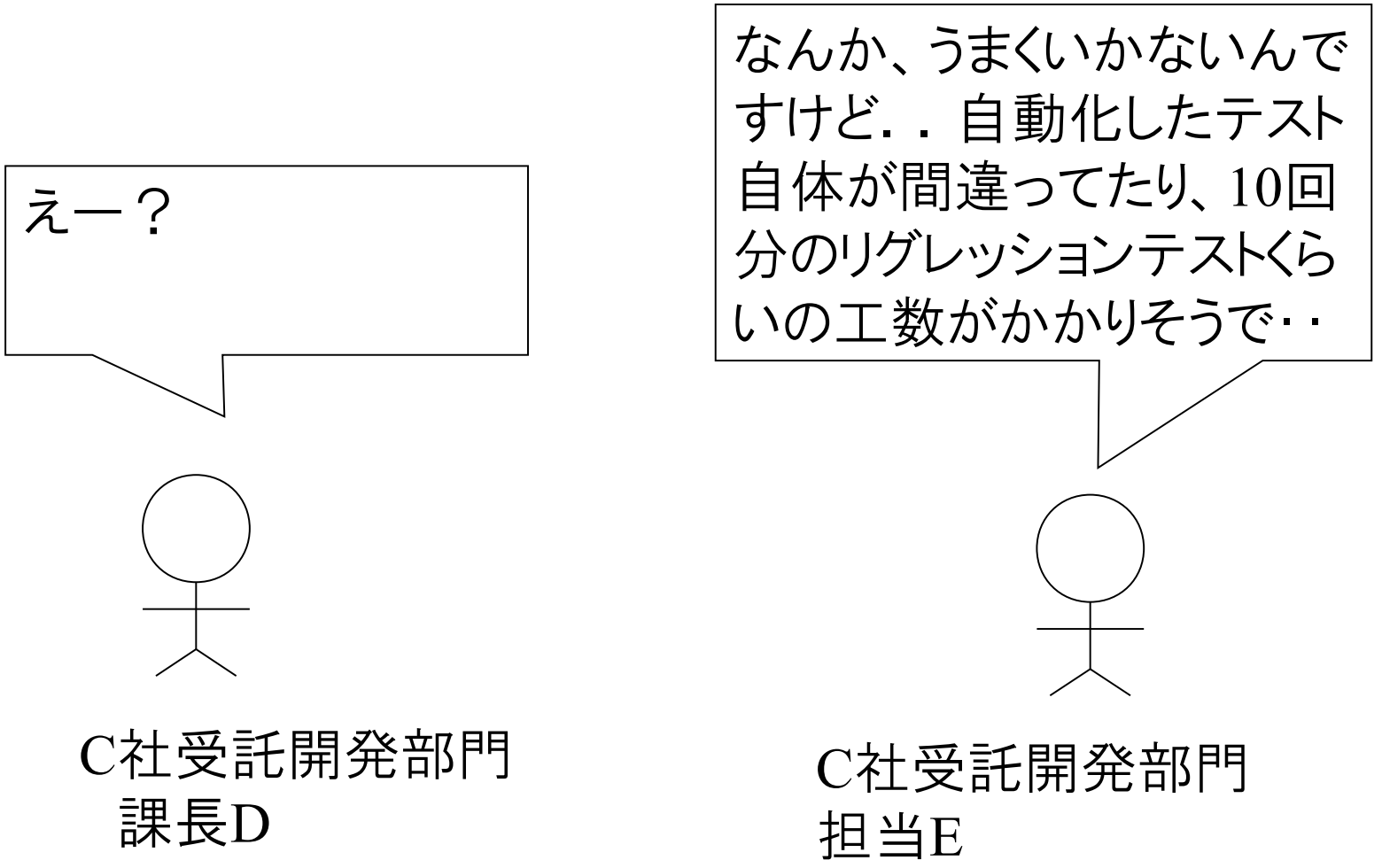
C社受託開発部門
課長D



C社受託開発部門
担当E

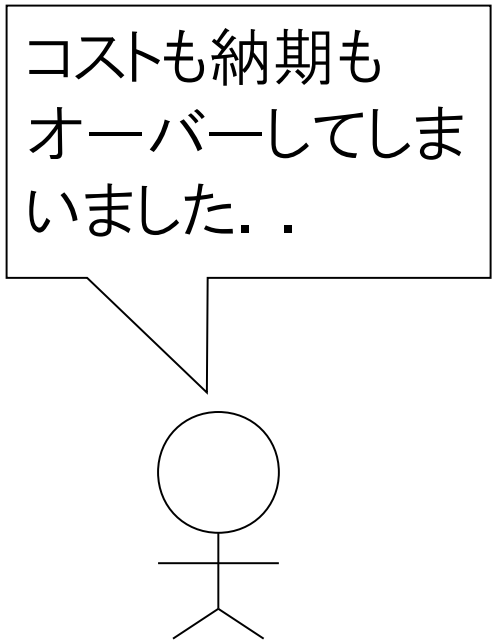
もう少しありそうな話を・・・

- 実際に開発をはじまってから・・・・

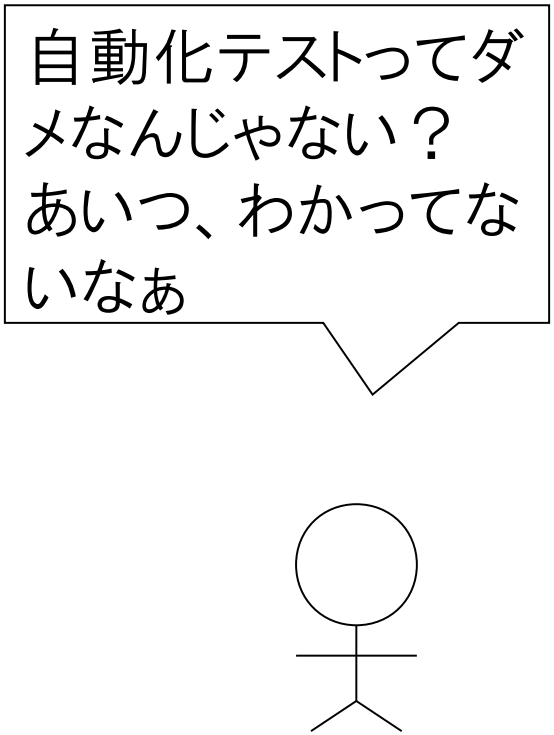


もう少しありそうな話を・・・

- 開発が終わり. . .



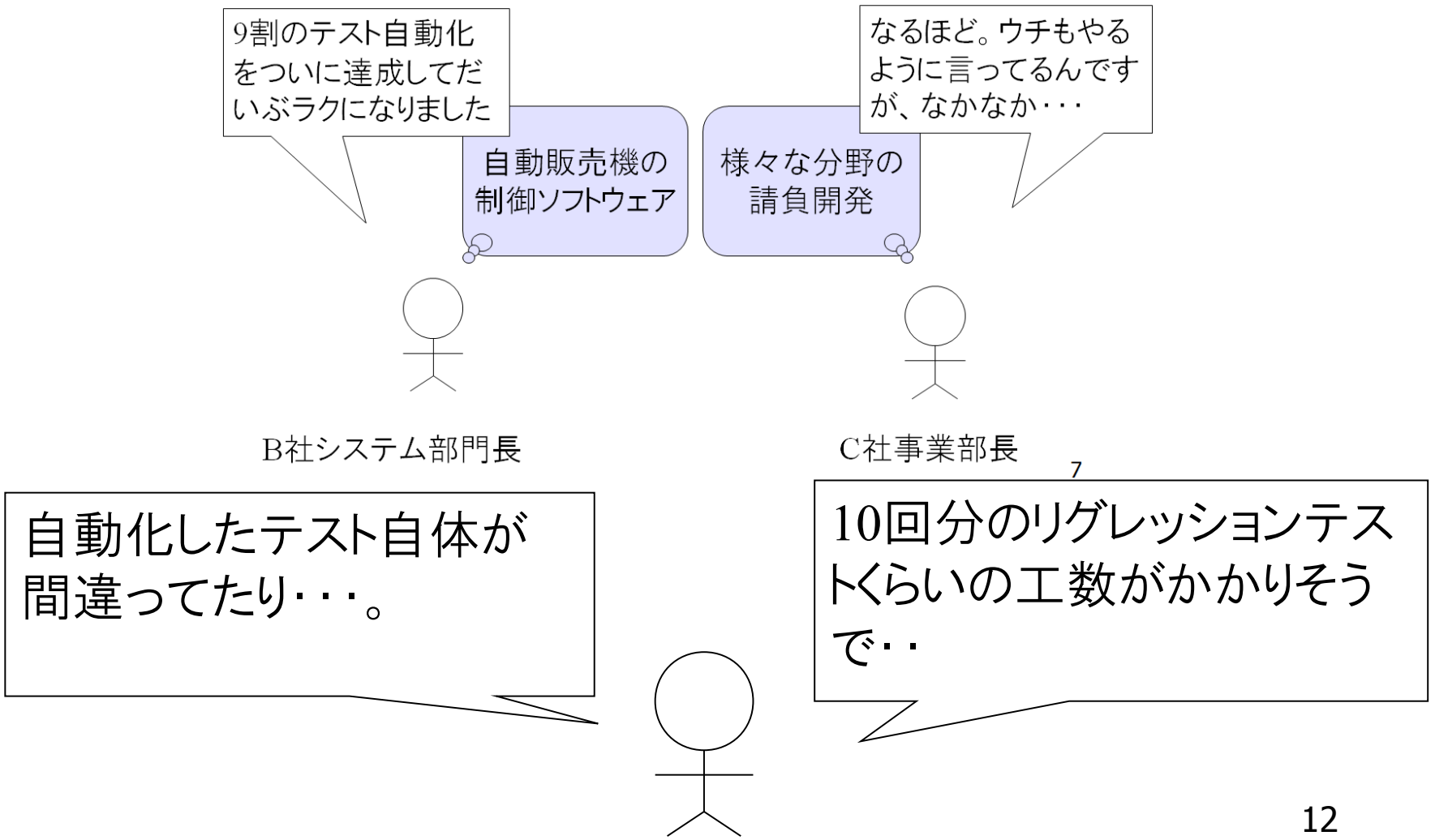
C社受託開発部門
課長D



C社事業部長

ここからコンテキストを抽出します

- どんな前提の違いが考えられるか？

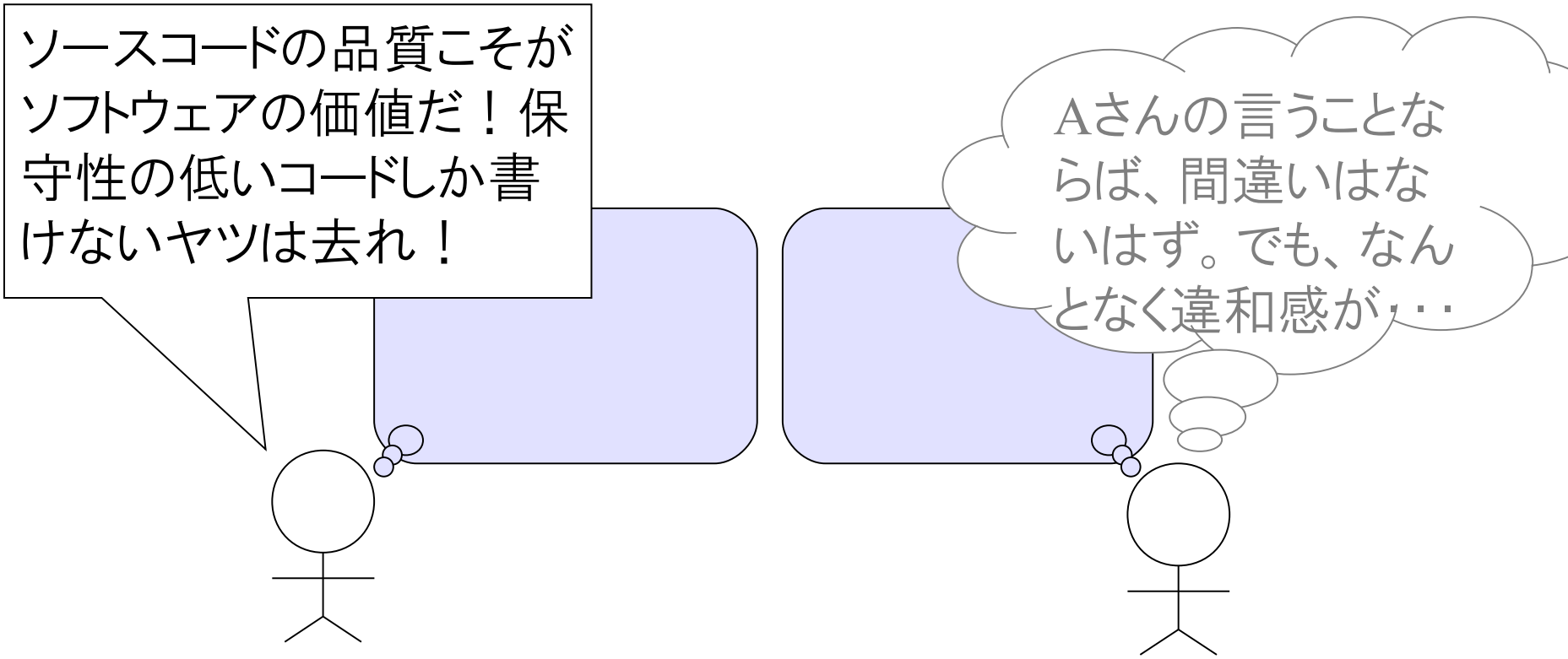


以降のあらまし

- コンテキストが明示された例
- コンテキスト
  - なぜコンテキストを明示するのか？
  - コンテキスト抽出の方針
- コンテキスト記述の例
  - サーベイ論文の紹介
  - 調査結果とコンテキストを記述した例
- 具体例とディスカッション

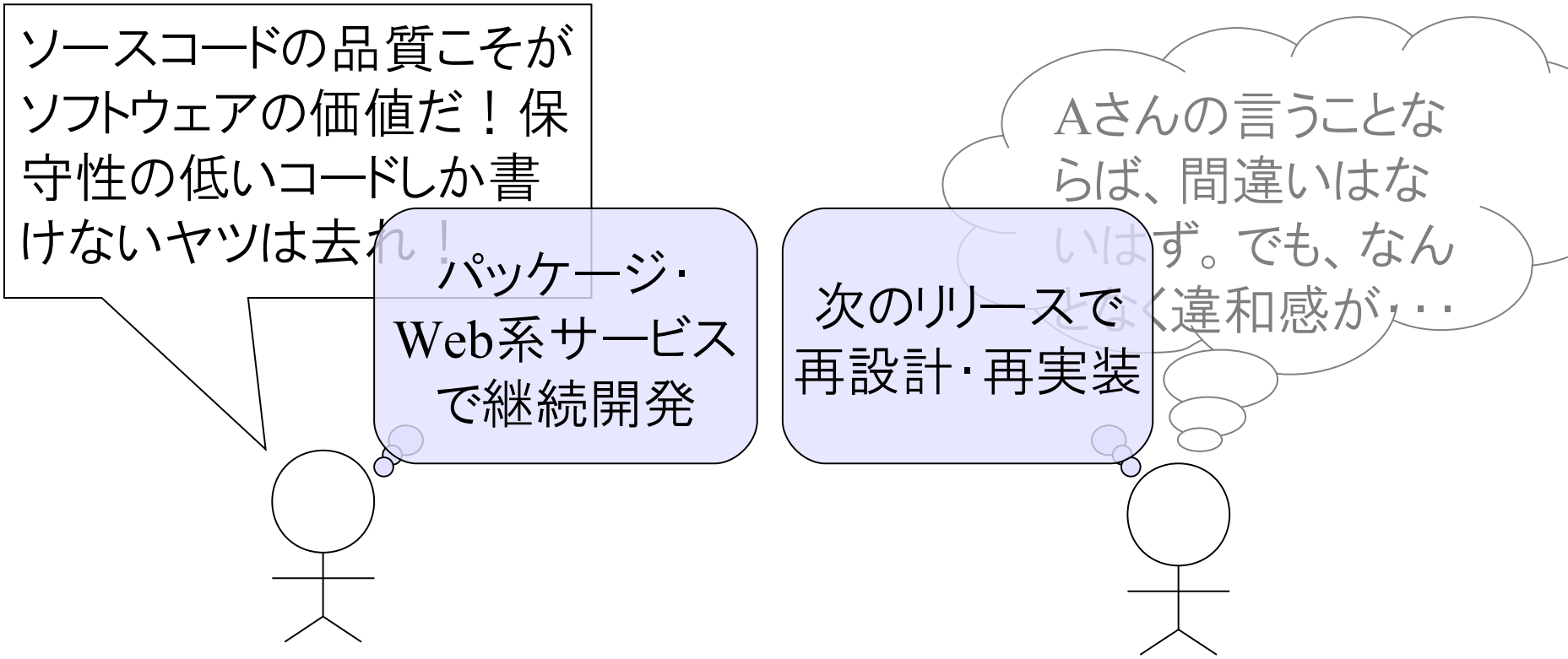
コンテキストを明示した例

- 質問: ソースコードの保守性をどう考えますか？



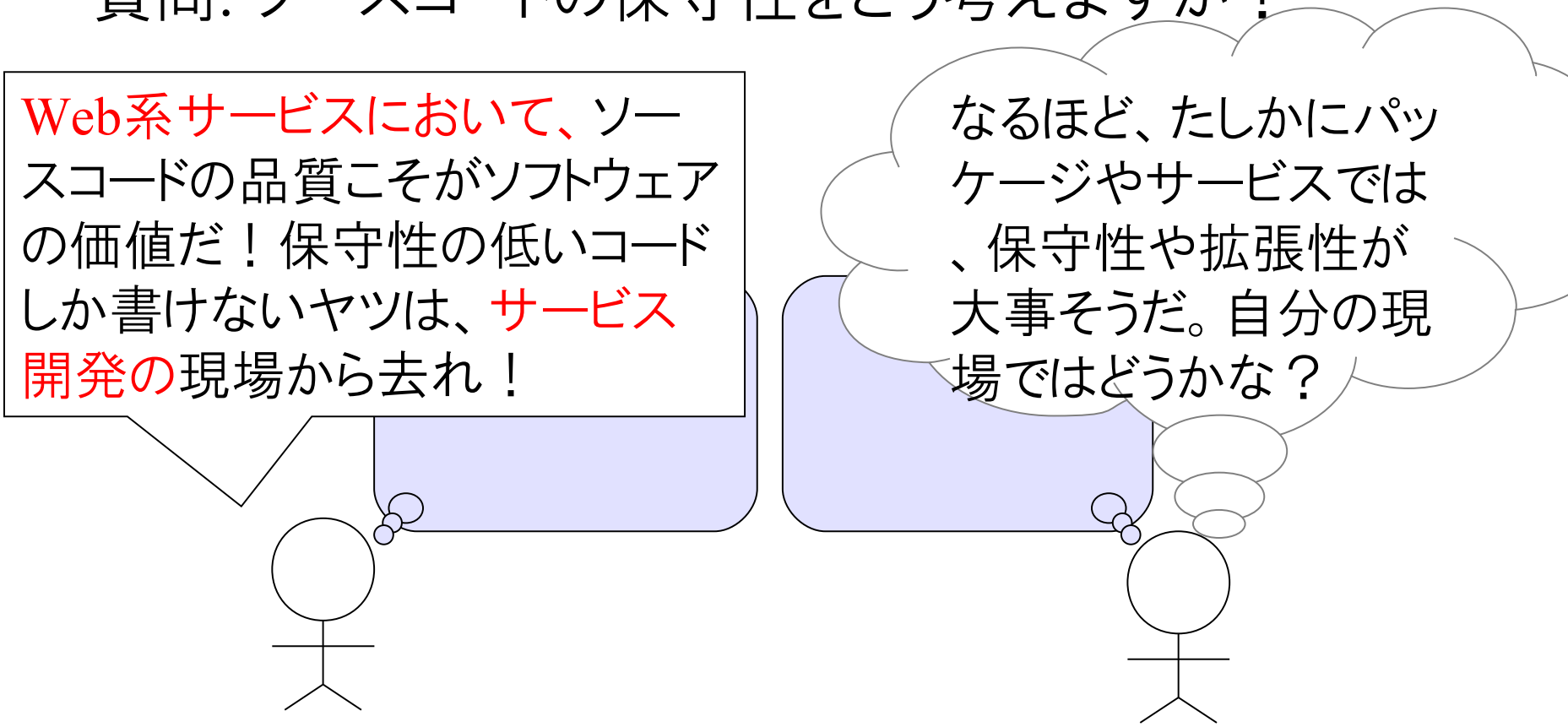
実は・・・・

- 質問: ソースコードの保守性をどう考えますか？



こう言ってくれたら・・・

- 質問: ソースコードの保守性をどう考えますか？



私がこの話をするコンテキスト

- サービス開発、SIに携わっていた。
  - サービス開発
    - 不特定多数のユーザを前提に定型サービスの開発
    - 想定ユーザ（ペルソナ）の設定をはじめ、実装、運用まで
  - システムインテグレーション
    - エンドユーザから受注し、基本設計よりも後を再発注
  - となりの部門では組込み用ソフトウェアを開発しており、そこでの話も漏れ聞いていた。
- ソフトウェア工学の研究者として、多くのソフトウェア開発企業と対話・相談している。
  - 6年間で300社、2011年は2500名の前で講演する予定で、フィードバックをいただいている。

コンテキスト抽出の動機

- 「なぜプラクティス、技術、技法を実際に適用して研究するのか？」Basili 教授の講演から

- プラクティス、技術、技法のユーザにとって
  - いつ使えばよいかが、わかる。
  - コンテキストにあっているかどうか確信を持てる。
- プラクティス、技術、技法の考案者にとって
  - 実現可能性を示せる。
  - 制限と制約を確認できる。
  - 今後の方向性や発展を考えることができる。

出典: V. Basili and S. Elbaum: Better Empirical Science for Software Engineering , 28th Presentation at International Conference on Software Engineering, http://www.cs.umd.edu/~basili/presentations/2006%20ICSE%20with%20Elbaum.pdf(2006)

エンピリカルアプローチの三段階

第一段階: 観察

- 実験や調査により現象を確認する。
- 現象が表現できる。

↓

第二段階: 法則の発見

- 現象が起こるコンテキストを理解する。
- 現象を予測できるようになる。

↓

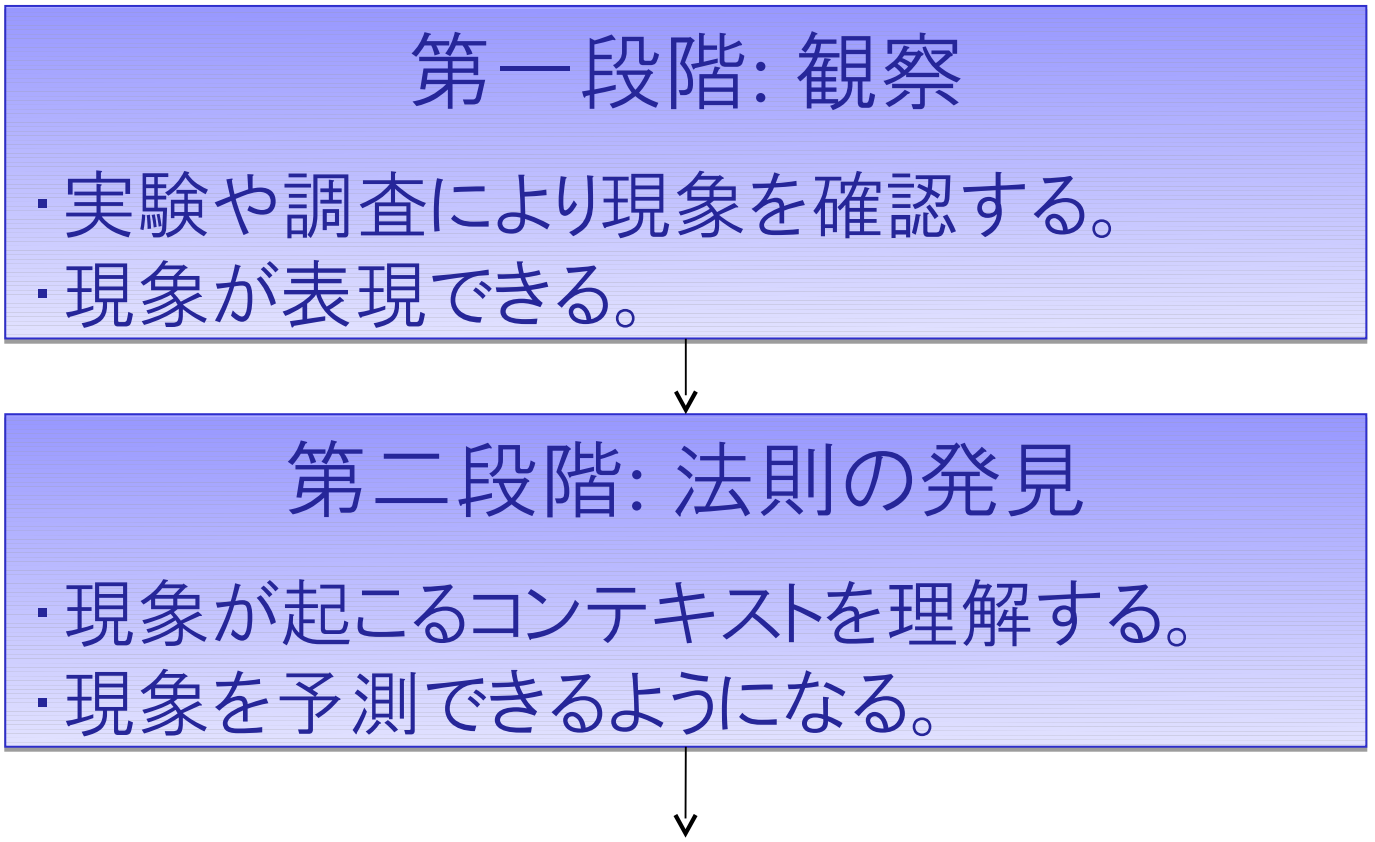
第三段階: 理論の確立

- 因果関係を明らかにする。
- 現象を説明できるようになる。

出典: ソフトウェア開発におけるエンピリカルアプローチ，アスキー，p.11(2008)

観察→コンテキスト

- コンテキストを決めてからではなく、観察や調査が先にある。



出典: ソフトウェア開発におけるエンピリカルアプローチ，アスキー，p.11(2008)

アジャイルソフトウェア開発のサーベイ

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Information and Software Technology 50 (2008) 833–859

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Empirical studies of agile software development: A systematic review

Tore Dybå *, Torgeir Dingsøyr

SINTEF ICT, S.P. Andersensv. 15B, NO-7465 Trondheim, Norway

Received 22 October 2007; received in revised form 22 January 2008; accepted 24 January 2008
Available online 2 February 2008

Abstract

Agile software development represents a major departure from traditional, plan-based approaches to software engineering. A systematic review of empirical studies of agile software development up to and including 2005 was conducted. The search strategy identified 1996 studies, of which 36 were identified as empirical studies. The studies were grouped into four themes: introduction and adoption, human and social factors, perceptions on agile methods, and comparative studies. The review investigates what is currently known about the benefits and limitations of, and the strength of evidence for, agile methods. Implications for research and practice are presented. The main implication for research is a need for more and better empirical studies of agile software development within a common research agenda. For the industrial readership, the review provides a map of findings, according to topic, that can be compared for relevance to their own settings and situations.

出典: T. Dyba and T. Dingsoyr: Empirical studies of agile software development: A systematic review, Journal of information and software technology, vol. 50, p. 833-839 (2008)

33本の論文の一覧表

Table 8
Quality assessment

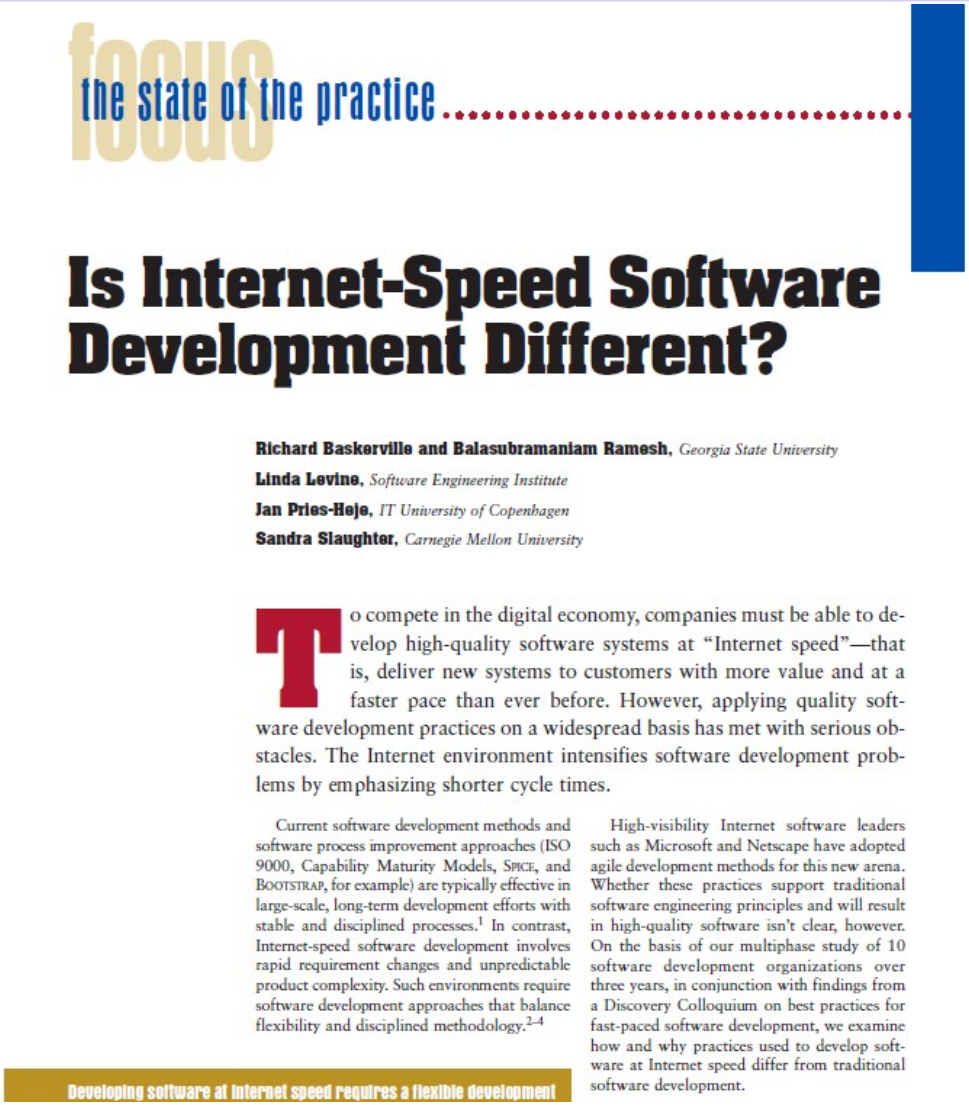
Study	1	2	3	4	5	6	7	8	9	10	11	Total
	Research	Aim	Context	R. design	Sampling	Ctrl. Grp	Data coll.	Data anal	Reflexivity	Findings	Value	
S1	1	1	1	0	0	0	1	1	0	1	1	7
S2	1	1	1	1	1	1	1	1	0	1	1	10
S3	1	1	1	1	0	0	0	0	0	1	1	6
S4	1	1	1	1	0	1	1	1	0	1	1	9
S5	1	1										
S6	1	1										
S7	1	1										
S8	1	1										
S9	1	1										
S10	1	0										
S11	1	1										
S12	1	1										
S13	1	1										
S14	1	1										
S15	1	1										
S16	1	1										
S17	1	1										
S18	1	1										
S19	1	1										
S20	1	1	1	1	0	0	1	1	0	1	1	8
S21	1	1	1	1	0	0	1	1	0	1	1	8
S22	1	1	1	1	1	0	1	1	0	1	1	9
S23	1	0	1	0	0	0	0	0	0	1	1	4
S24	1	1	1	1	0	0	1	1	0	1	1	8
S25	1	1	1	1	0	0	0	0	0	1	1	6
S26	1	1	1	1	0	0	0	0	0	1	1	6
S27	1	1	1	1	0	0	1	1	0	1	1	8
S28	1	1	1	1	1	1	1	1	0	1	1	10
S29	1	1	1	1	1	0	1	1	0	1	1	9
S30	1	1	1	1	1	0	1	1	0	1	1	9
S31	1	1	1	0	0	0	0	0	1	1	1	6
S32	1	1	1	1	0	1	1	1	0	1	1	9
S33	1	1	1	1	0	1	1	1	0	1	1	9
Total	33	31	33	30	8	10	26	25	1	33	33	

Table 8
Quality assessment

Study	1	2	3	4	5
	Research	Aim	Context	R. design	Sampling
S1	1	1	1	0	0

出典: T. Dyba and T. Dingsoyr: Empirical studies of agile software development: A systematic review, Journal of information and software technology, vol. 50, p. 833-839 (2008)

“インターネットスピード”の開発というコンテキスト



focus
the state of the practice

Is Internet-Speed Software Development Different?

Richard Baskerville and Balasubramaniam Ramesh, *Georgia State University*
Linda Levine, *Software Engineering Institute*
Jan Pries-Heje, *IT University of Copenhagen*
Sandra Slaughter, *Carnegie Mellon University*

To compete in the digital economy, companies must be able to develop high-quality software systems at “Internet speed”—that is, deliver new systems to customers with more value and at a faster pace than ever before. However, applying quality software development practices on a widespread basis has met with serious obstacles. The Internet environment intensifies software development problems by emphasizing shorter cycle times.

Current software development methods and software process improvement approaches (ISO 9000, Capability Maturity Models, SPICE, and BOOTSTRAP, for example) are typically effective in large-scale, long-term development efforts with stable and disciplined processes.[1] In contrast, Internet-speed software development involves rapid requirement changes and unpredictable product complexity. Such environments require software development approaches that balance flexibility and disciplined methodology.[2–4]

High-visibility Internet software leaders such as Microsoft and Netscape have adopted agile development methods for this new arena. Whether these practices support traditional software engineering principles and will result in high-quality software isn’t clear, however. On the basis of our multiphase study of 10 software development organizations over three years, in conjunction with findings from a Discovery Colloquium on best practices for fast-paced software development, we examine how and why practices used to develop software at Internet speed differ from traditional software development.

Developing software at Internet speed requires a flexible development

出典: R. Baskerville, B. Ramesh, L. Levine and S. Slaughter: Is Internet-Speed Software Development Different?, IEEE Software, vol. 20, no. 16, p70-77(2003)

“インターネットスピード”の開発というコンテキスト

Software development involves achieving interoperability and integration among components developed elsewhere or purchased off the shelf.

surance all occur simultaneously, but sequentially on different releases. Sometimes developers begin coding even before they fully understand a project's requirements. Development proceeds in anticipation of the features that will be required in the product's final version.

Release more often

"People have a perception of Internet speed. They expect it. So we've had to scope our delivery or deliver a smaller set of features, thereby releasing more often," said one manager we interviewed. Requirements can stay fluid when a company constantly monitors and prioritizes the features that will be included in successive releases of the product. Features that can't be completed in time can slip from one release to the next. Similarly, the company can introduce an important new feature rather late in the process when market conditions require it. The fast cycle time removes much of the trauma of slipping a feature.

Depend on tools

Many of the companies we visited make heavy use of development tools and environments that speed the design and coding process. The infrastructure and tools provided by new technologies offer much of the functionality that used to be custom-built in traditional software development. One developer estimated that "50 percent of development is already taken care of by the tools we use."

Implant customers in the development environment

Customer involvement in traditional software development is usually through reference groups or steering committees, which can slow development. When requirements are fuzzy and fast-changing, intimate access to customers slashes time delays and ensures that high-priority features are correctly identified. Customers move their normal working environment to the development environment to shorten cycle times. This "implantation" lets customers participate closely in all phases of development. Internet projects rely on this close involvement rather than a formalized requirements management process. It provides customers with immediate feedback on the costs and schedule implications of requirements changes and leads to

Establish a stable architecture

A fixed architecture helps anchor a rapid development process, which is never quite stable. It allows similarity among releases and the largest possible reuse of components. A three-layer architecture—database, business logic, and user interface—is common.

Some companies we interviewed focused on developing a common architecture and standardizing it across applications. They viewed this as an investment that would pay huge dividends by facilitating development of scalable and maintainable systems. Other companies did not apply this practice because throwaway systems might not need stable architectures to operate. The effects of bad architecture can emerge far downstream in the product evolution. By that time, the product might be undergoing a complete redevelopment.

Assemble and reuse components

Internet-speed development can be achieved often only if developers maximize reusable components, rather than craft the software from scratch. "Internet speed needs reuse. We need to take components or assets and know how to put them together," one developer said during our interview. A manager from the same company continued, "The strategy is to acquire, integrate, and assemble components with wrappers—to get things done quickly."

Component reuse at all levels of the architecture (business logic, interfaces, and back-end infrastructure) was prominent among companies interviewed. Software development involves achieving interoperability and integration among components developed elsewhere or purchased off the shelf.

Ignore maintenance

The short life span of Internet-speed software means that developers rarely give maintenance serious consideration. One small software house said, "Products are not documented: no design document, no requirements specification. If the person who originally did it is gone, it takes a much longer time. Often we can start from scratch. It leads to a throw-away mentality." When software is retired quickly and replaced with newer versions developed from scratch, this cavalier attitude might avoid the serious maintenance

Release more often

"People have a perception of Internet speed. They expect it. So we've had to scope our delivery or deliver a smaller set of features,

コンテキスト

事実（調査結果）

出典: R. Baskerville, B. Ramesh, L. Levine and S. Slaughter: Is Internet-Speed Software Development Different?, IEEE Software, vol. 20, no. 16, p70-77(2003)

当該記事で記述されている事実とコンテキスト

- 頻繁なリリース
  - 事実: スコープを小さくして頻繁なリリースをする。
  - コンテキスト: 市場に応じて新しい機能をリリースしなければならない。
- オンサイト顧客
  - 事実: 顧客は通常の自席から開発ルームにすぐ移動できるようにする。
  - コンテキスト: 戦略が適宜変更するためにユーザ要求が頻繁に変わる。
- 保守性の優先度を下げる(保守性を無視する)
  - 事実: 設計ドキュメントはない。
  - コンテキスト: すぐに作り替えられるので保守性は問題になりにくい。

出典: R. Baskerville, B. Ramesh, L. Levine and S. Slaughter: Is Internet-Speed Software Development Different?, IEEE Software, vol. 20, no. 16, p70-77(2003)

その他のコンテキストの例

- 品質のプライオリティが高い。
- 性能を満たすことが至上命題
- コア部分のリリース日は絶対守らなければならない。
- 法令遵守が必須（でも、法令はまだ決まってない）
- 類似製品・サービスよりも早くリリースできることが、もっとも重要
- 複数拠点
- 再委託している。
- ユーザ側の体制

ここでお題を

- ご自身が携わっている開発で・・・

- 「ソフトウェア品質」とはどのように位置づけられ
- そのためにテストにおいて、どのような施策・方法がとられていますか？

  そして

- それによってどのような競争力を生み出そうとしていますか？

ご自身の開発でコンテキスト抽出してみてください

ソフトウェア品質は (a)＿＿＿＿＿＿＿＿と位置づけられ
(b)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿が優先される。
そのため、テストでは(c)＿＿＿＿＿＿＿＿＿＿＿＿＿＿ 。
その理由は(d)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿に
よって競争力が生み出されると考えられているから。

(a)
(b)
(c)
(d)

一例（高品質）

- ソフトウェア品質は システム全体を安定して稼働するための制御の中枢を担うもの と位置づけられ 期待通りに動作し、どのような利用者にとっても安心して利用できることが優先される。
  そのため、テストでは 網羅的な組合せテスト、様々な環境でのテスト、複数の利用者の想定によるシナリオテストが用意されている。
  その理由は 全ての顧客が、システムに安心・安全を求めており、利用環境が異なる様々な顧客層を包括的な単一システムで安価に提供すること によって競争力が生み出されると考えられているから。

一例（UX: User eXperience design）

- ソフトウェア品質は 製品の競争力（特に魅力品質）と位置づけられ 利用者に初めての感覚や驚きを与えられるかどうかが優先される。
  そのため、テストでは フィールドテストや一般ユーザによる評価と作り直しのためのイテレーションが用意されている。
  その理由は 多くの顧客は初めての感覚や驚きを製品に求めており、その評価によって新製品を手に取ってくれたり、他製品への乗り換えを検討したりしないことによって競争力が生み出されると考えられているから。

ウォータフォール開発の前提: Publickeyの記事

ウォーターフォールだって成功する。ただしそれには前提条件があるはず

2011年1月28日

東京証券取引所の新株式売買システムの刷新という大規模な開発プロジェクトの背景と、成功要因を、東証の担当者自身が説明したプレゼンテーションを記事にした「客が本気にならないといいシステムができない。東証arrowhead成功の鍵とは ～ Innovation Sprint 2011」は公開から3日後の現在、過去1週間でもっとも読まれた記事の第1位になっており、たくさんの反響をいただきました。

記事で取り上げたarrowheadの開発は巨大なウォーターフォール型のプロジェクトであり、その成功の鍵として主に挙げられていたのは次のようなことでした。

- 関係者全員での危機意識の共有
- 発注者責任の明確化
- 上流工程完璧主義

客が本気にならないといいシステムができない。東証arrowhead成功の鍵とは ～ Innovation Sprint 2011 – Publickey

しかしこの成功の鍵には、語られなかったけれども理解しなく

Google™カスタム検索 検索

Blogger in Chief

Junichi Niino(jniino)
IT系の雑誌編集者、オンラインメディア発行人を経て独立。新しいオンラインメディアの可能性を追求しています。
(詳しいプロフィール)

Publickeyの新着情報をチェックしませんか？
Twitterで：@Publickey
RSSリーダーで：Feed
≫ 過去の記事を読む

▼ Publickeyスポンサー広告

出典: http://www.publickey1.jp/blog/11/post_141.html（2011/1）

今日のお話を文書化したもの

- Software Testing ManiaX vol. 4
  - Clarifying the context of your methods, approaches, insights, implications and case studies
- EMWest vol. 2
  - ソフトウェアレビューの価値

- Web
  - 「なぜウチではうまくいかないか？」を考える開発コンテキストの解説？
    http://blogs.itmedia.co.jp/morisaki/2010/07/post-0672.html

まとめ

- コンテキストが一致していない話の例を紹介した。
  - スマートフォンとECU
- コンテキストを明示した例を紹介した。
  - 保守性
- コンテキストの意義
  - コンテキスト設定までのステップ
- 事実（観察結果）とコンテキストの例
  - 論文: Is Internet-Speed Software Development Different?
- 具体例を題材にコンテキストを考えた。
- 本シンポジウムのSIGでぜひ活かしていただきたいです！

お知らせ

1. 論文、記事等の公開情報をお知らせするメールニュースご希望の方はメールアドレスをお知らせください。
   - From: 情報をお送りするメールアドレス
   - To: ismoris@ipc.shizuoka.ac.jp
   - Subject: 論文、記事等の公開情報のお知らせ希望
   - 本文
     ご氏名:
     ご所属:
   - いただいた情報を本来の目的以外で使用することはありません。
1. 当研究グループとの具体的な連携（有償の受託・共同研究）を募集しています。
   - 詳細な統計データや海外動向との比較ができます。
   - 東海・関西・首都圏の拠点の企業とも多く連携しています。