がんばるだけの品質向上活動からの脱却

森崎　修司

静岡大学 情報学部

http://twitter.com/smorisaki/

本資料に含まれる研究の一部は文部科学省「次世代IT基盤構築のための研究開発」の委託に基づいて行われたものです。また、本資料に含まれる研究の一部は同省科学研究補助費（若手B：課題番号21700033）の助成を受けました。

自己紹介

- 奈良先端科学技術大学院大学 博士後期課程 修了（2001.3）
  - ソフトウェアの利用品質
- エンジニアとして: ソフトウェア開発（2001～）
  - オンラインストレージサービスの開発、開発管理
  - 無線ICタグの研究開発（国際標準策定、ソリューション化）
  - コストカット部門、プリセールス
- 研究者として: ソフトウェア工学の研究（2005～）
  - 文科省e-society基盤の総合開発、経産省 先進的ソフトウェア開発でソフトウェア工学の産学連携の研究に従事
  - 300社超との相談、18社と機密保持契約ベースの産学連携
  - @IT, 日経SYSTEMS, ThinkIT等での連載記事
- 好きなこと
  - 講演冒頭等で失敗するかもしれないアグレッシブなネタを仕込んでおく姿勢

品質向上活動の5つのレベル

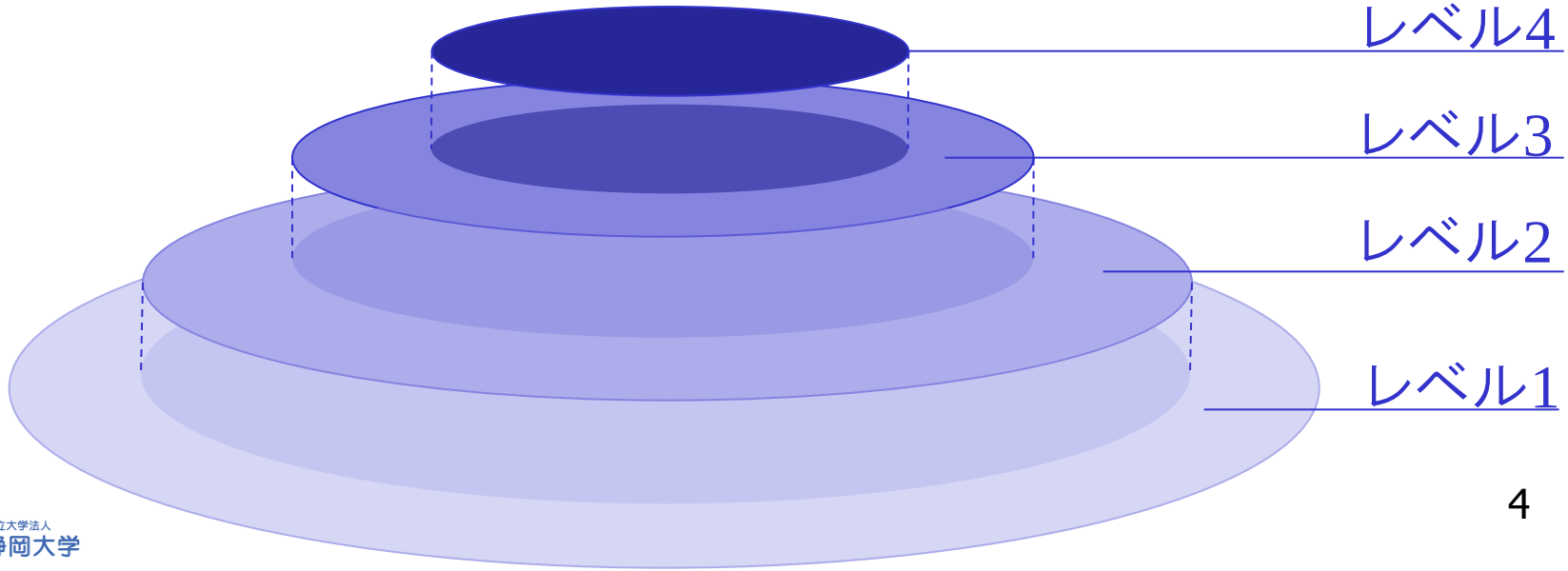
0. 品質向上とは関係ない仕事をしている
1. 個々人が思い思いの品質向上活動をしている
2. 関係者が合意した上で品質向上活動ができている
3. 2のうち、他組織ではマネが難しいものがわかっている。
4. 3のうち競争力の源泉となっているものがわかっている。

- ここでの「品質向上活動」は品質向上を目指した活動であり、議論の簡便化のため、必ずしも結果は伴っていなくてもよいものとする。

5つのレベル

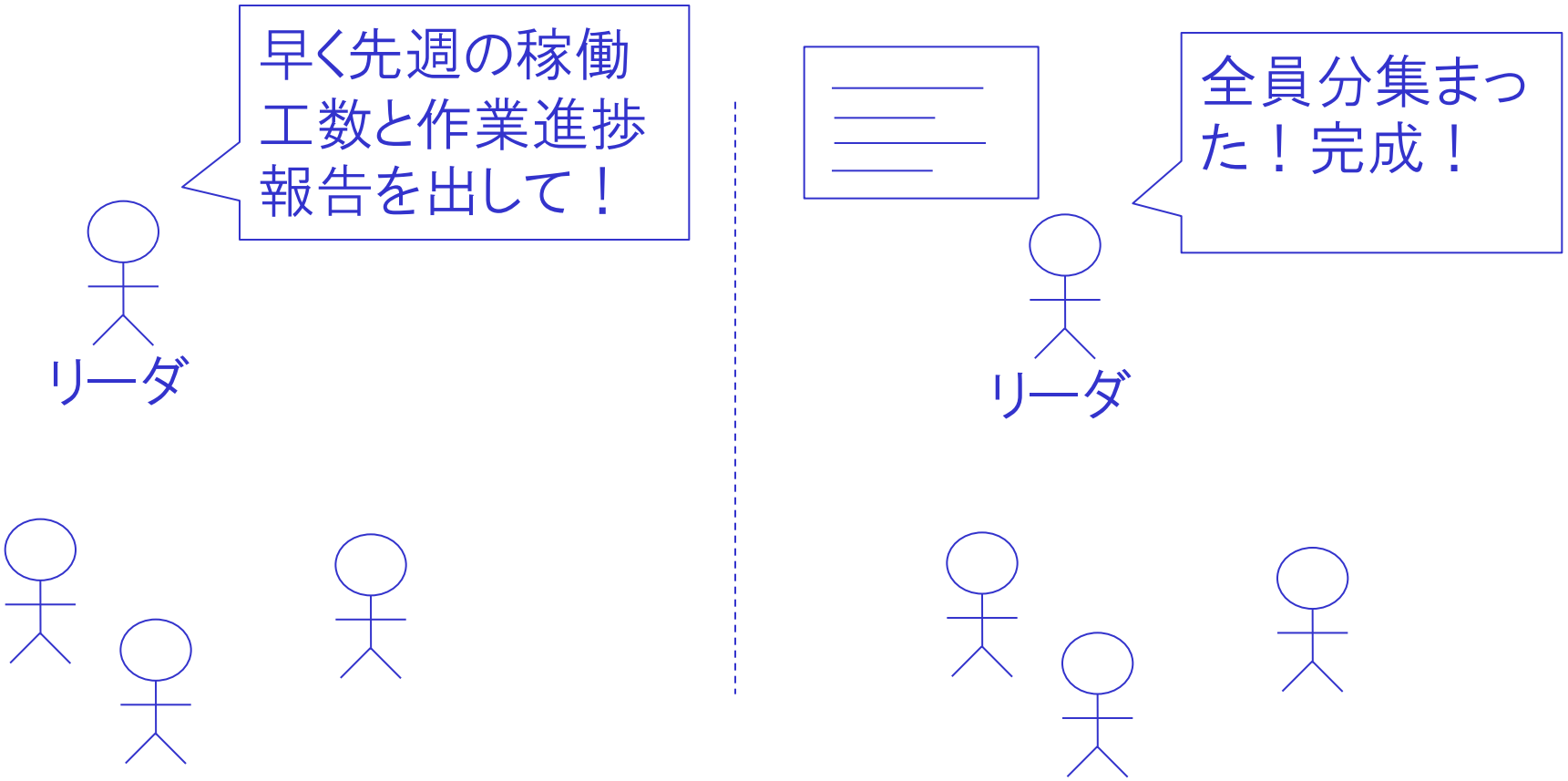
- 0. 品質向上とは関係ない仕事をしている。
- 1. 個々人が思い思いの品質向上活動をしている。
- 2. 関係者が合意した上で品質向上活動ができている。

- 3. 2のうち品質向上活動が特徴づけられているものがわかっている。
- 4. 3のうち、競争力の源泉となっているものがわかっている。



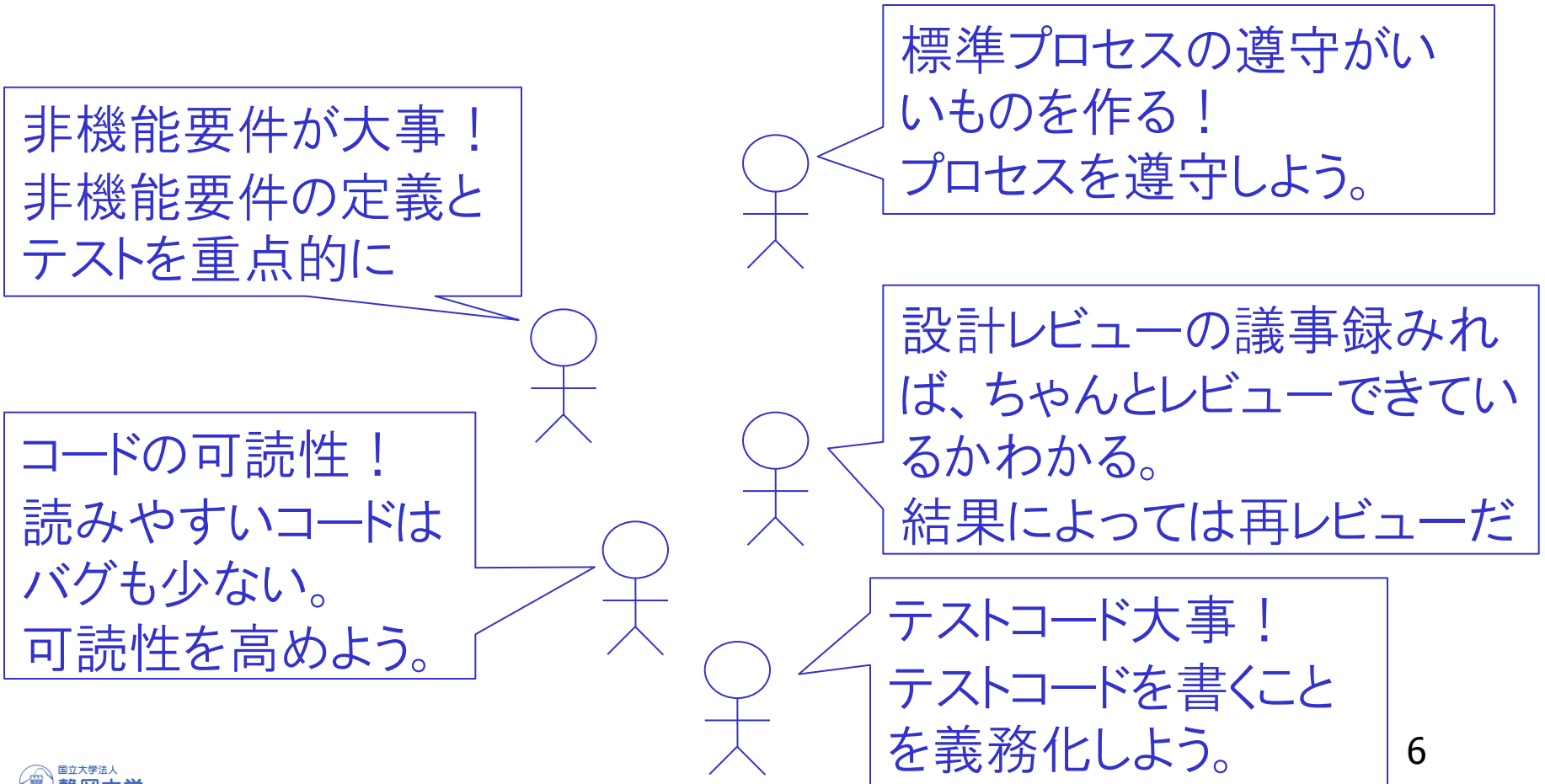
レベル0 – 品質向上とは関係ない活動をしている

- 品質向上につながる活動をしているようにも見えるが実際には効果のない活動をしている。



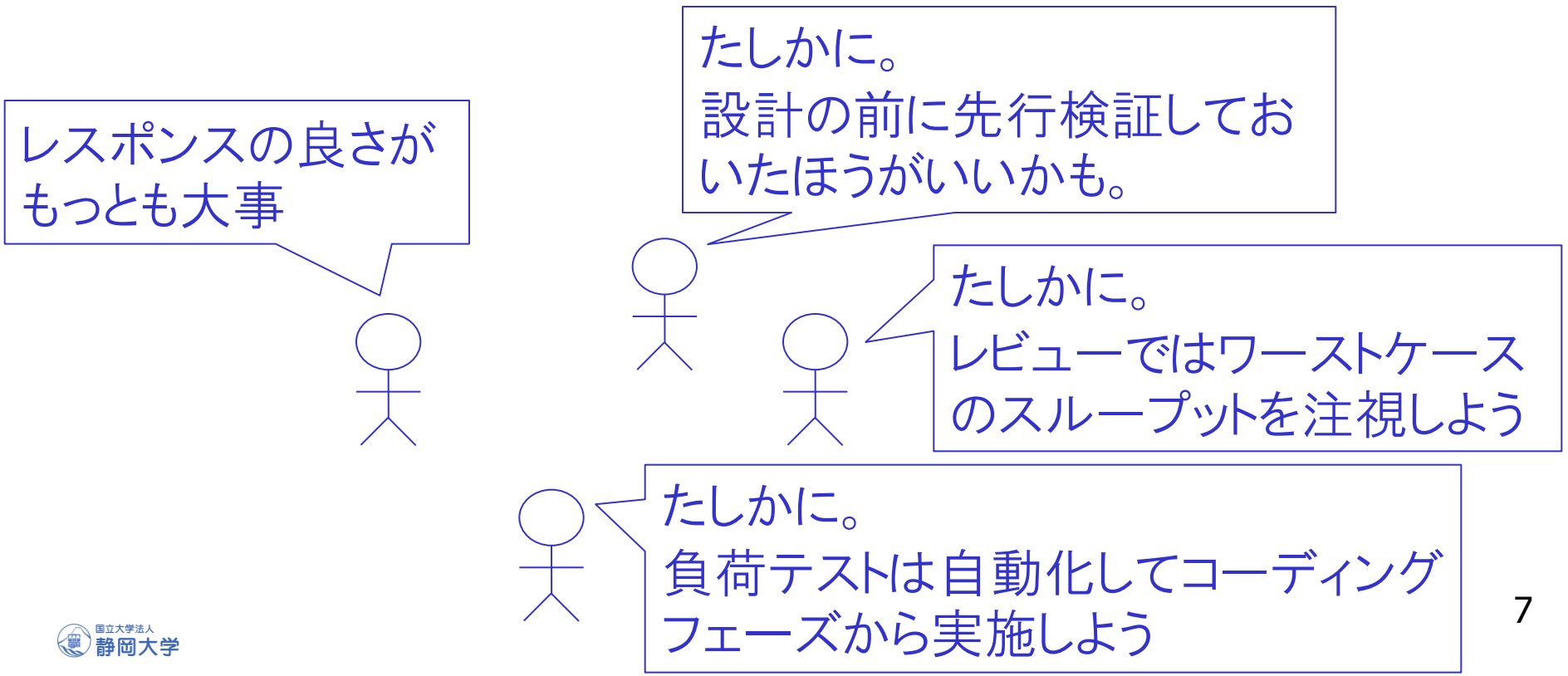
レベル1 – 思い思いの品質向上活動をしている

- 各々が品質向上につながると考え、活動をしているが合意・統一されていない。



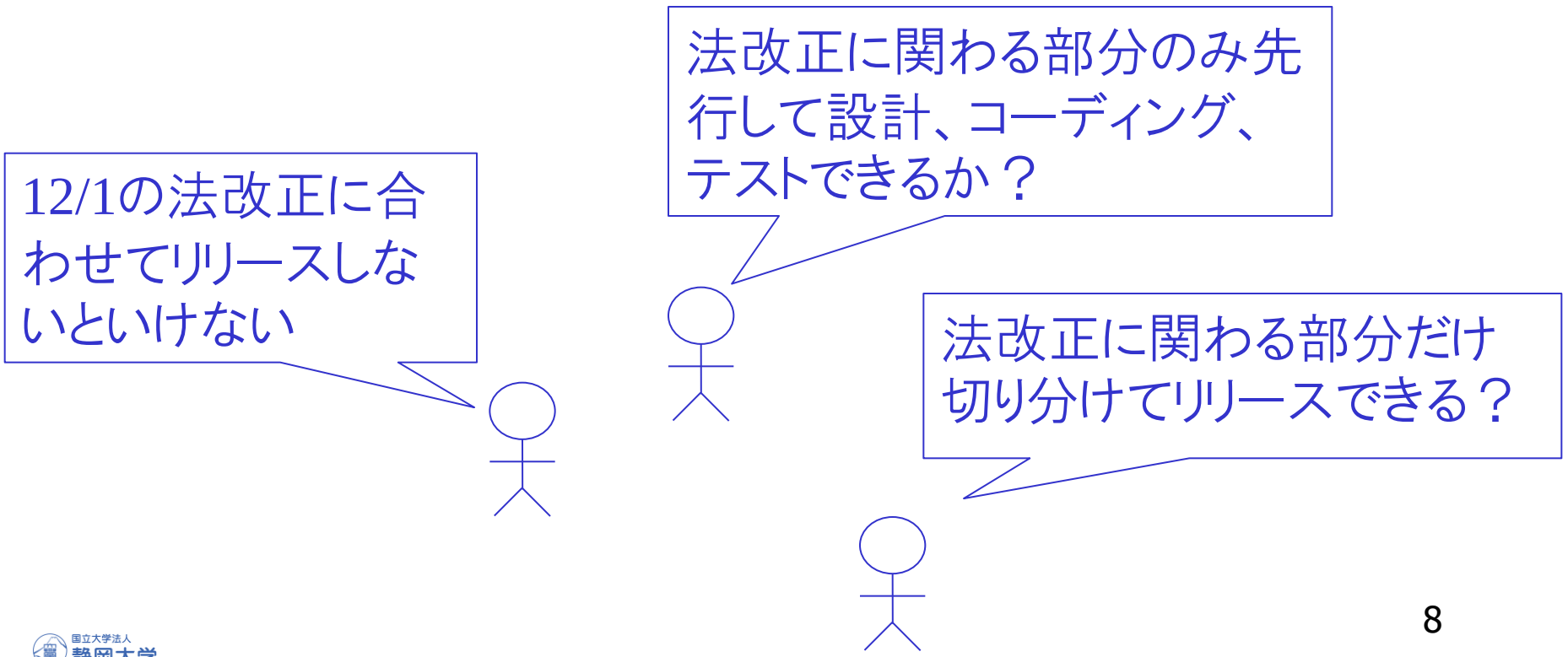
レベル2 – 合意している(1／2)

- 関係者の間で求められる品質が合意できている。
- 合意できた品質の向上に役立つ活動ができている。



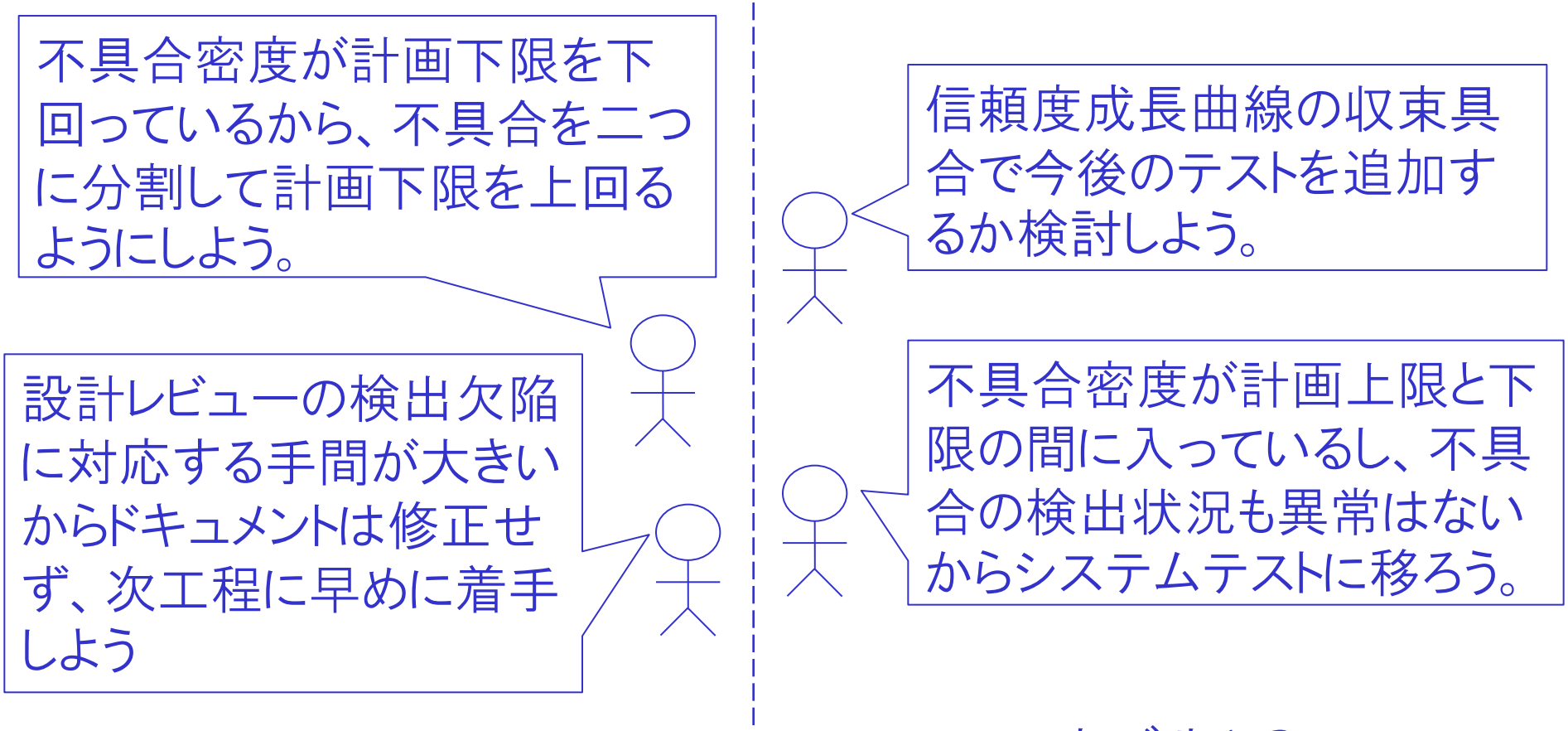
レベル2 – 合意している(2／2)

- 関係者の間で求められる品質が合意できている。
- 合意できた品質の向上活動ができている。



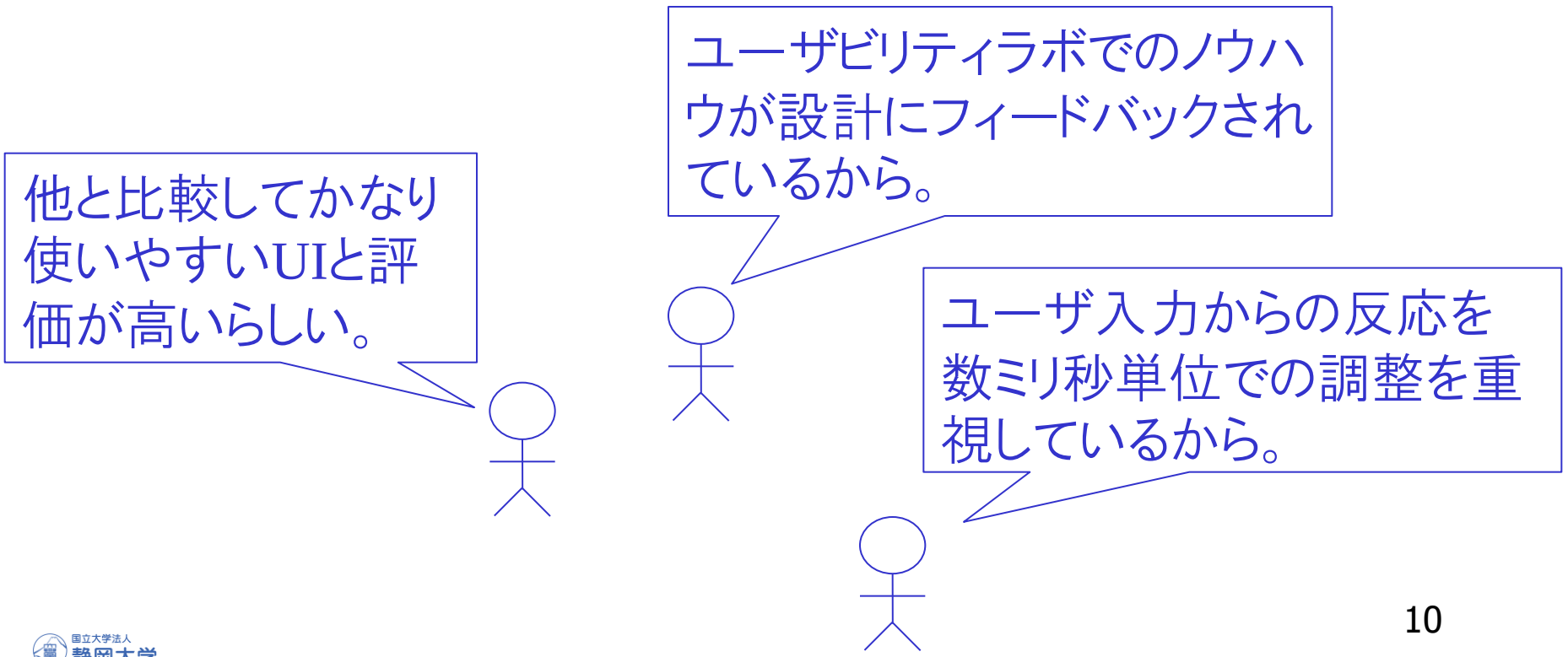
レベル0と1、2の違い

- 厳しめに見れば様々なものがレベル0とできるが、明らかにずれた活動でなければレベル1、レベル2とする。



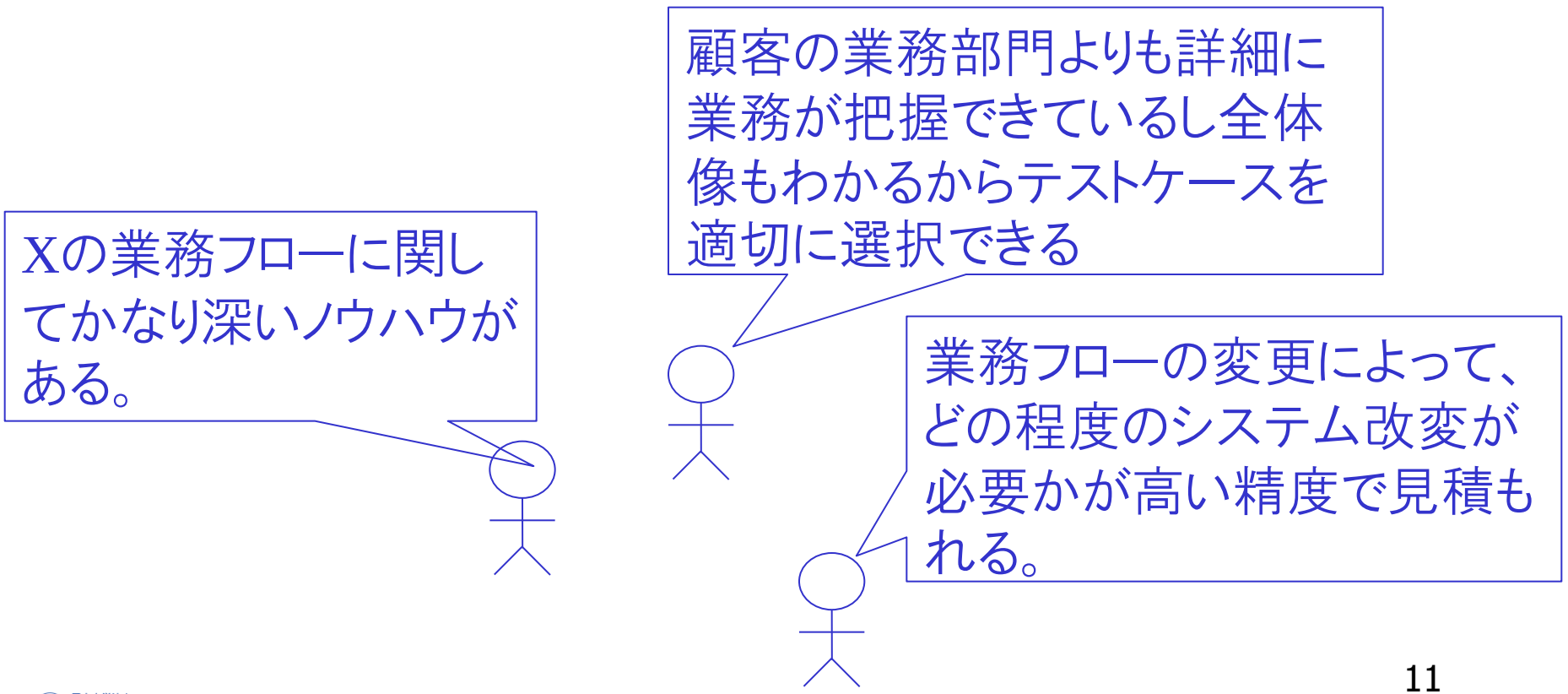
レベル3 – 特徴付けられている

- どのような活動によって品質向上が実現されているかがわかっており、品質向上活動の効果が出ている。
- 競合となり得る他の組織では簡単に模倣することが難しい。



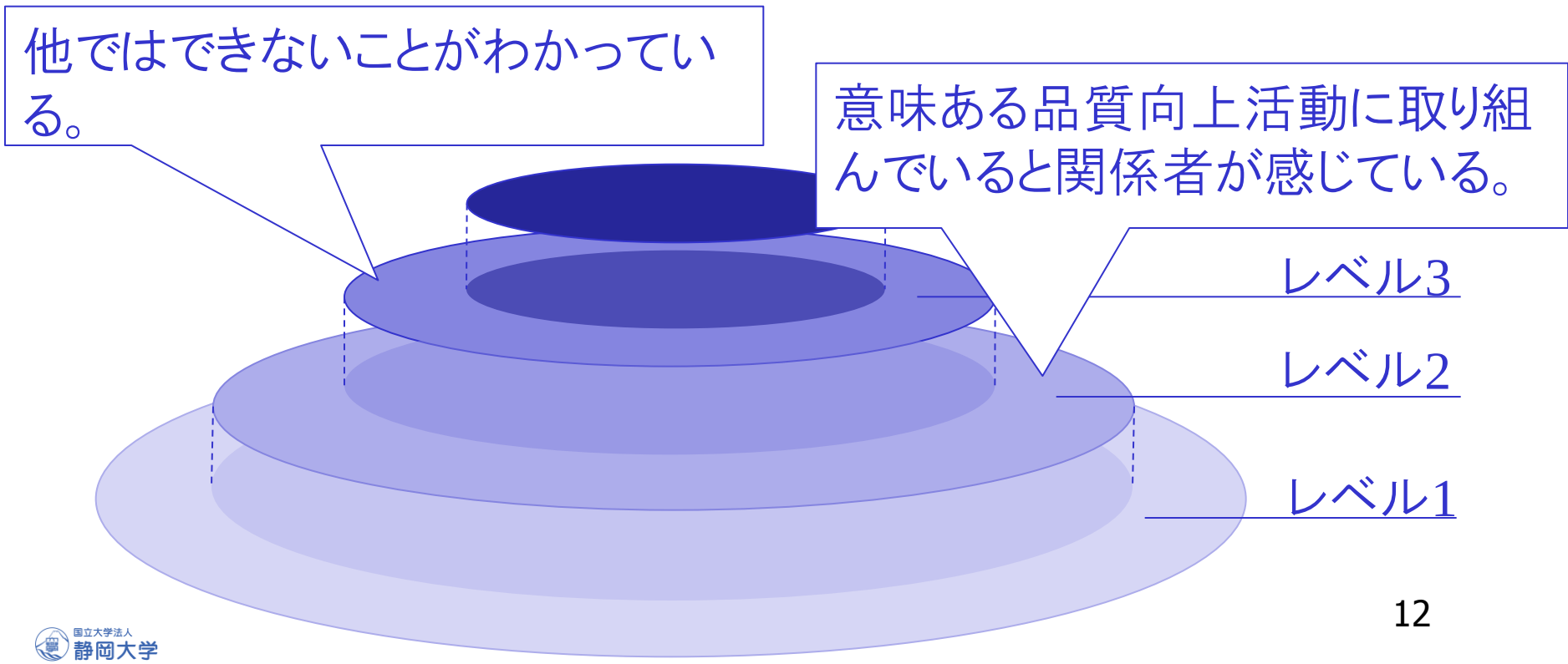
レベル3 – 特徴付けられている

- どのような活動によって品質向上が実現されているかがわかっており、品質向上活動の効果が出ている。
- 競合となり得る他の組織では簡単に模倣することが難しい。



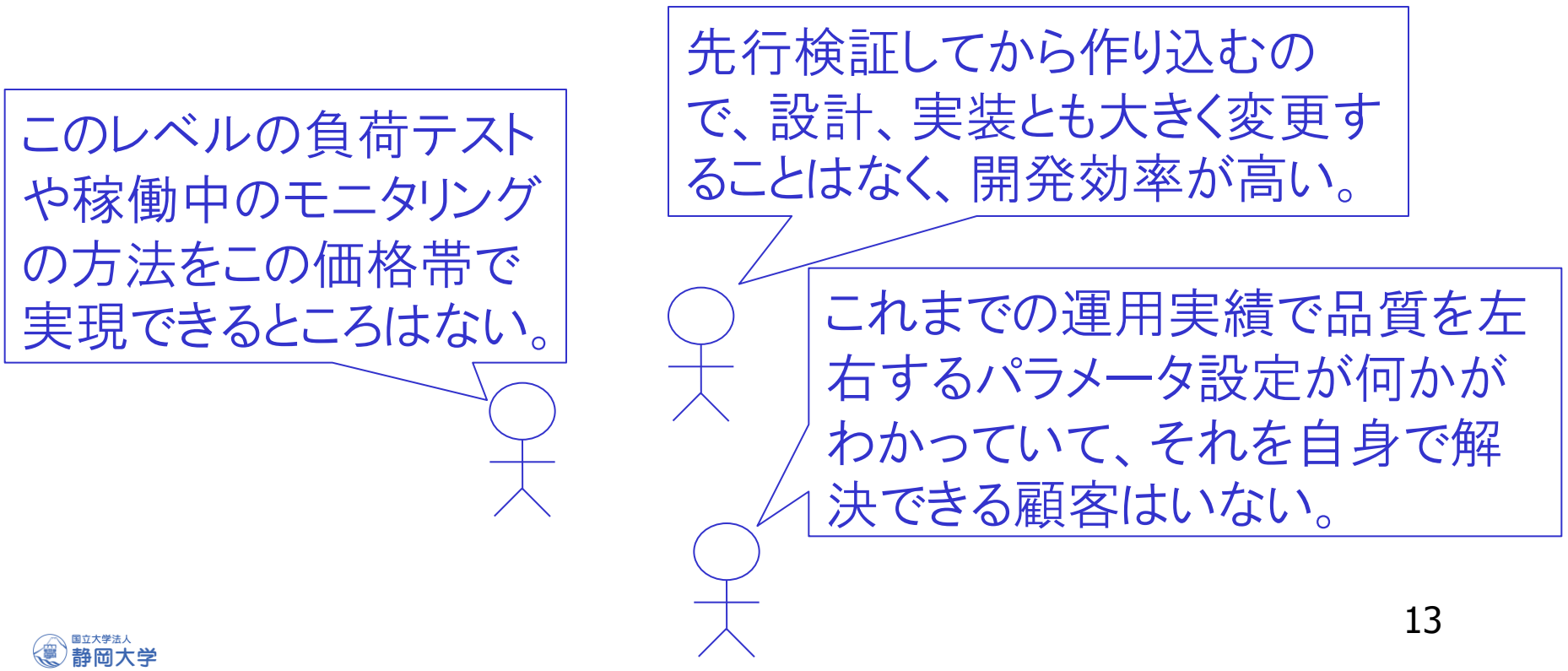
レベル2と3の違い

- 2 関係者が合意した上で品質保証活動ができている
- 3 品質保証活動が特徴付けられている



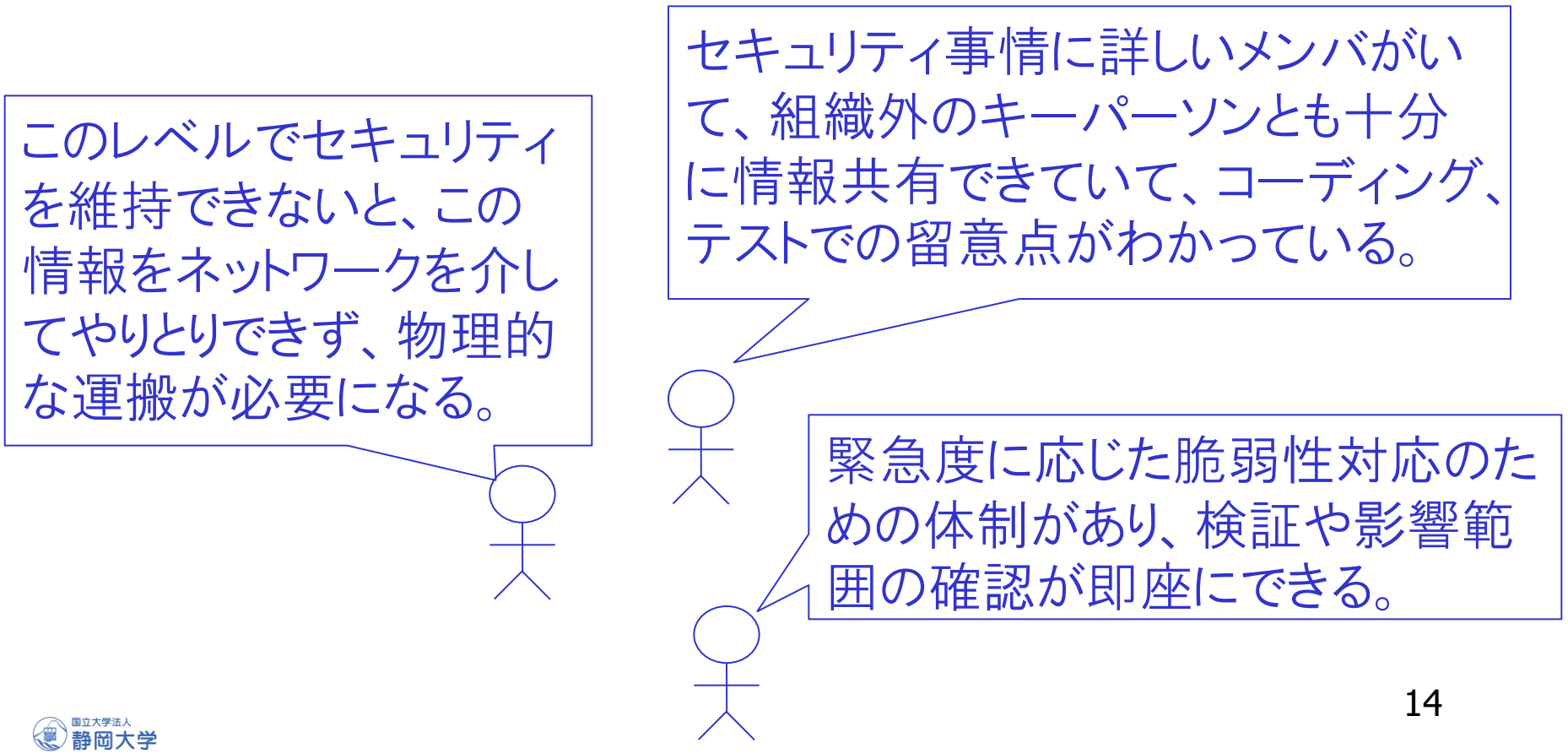
レベル4 – 競争力の源泉となっている(1／2)

- 品質が競争力の源泉となっており、その品質に対して顧客やユーザが価値を認めて対価を支払っている。
- 特定のソフトウェア、サービス、製品のプライスリーダになることができる。



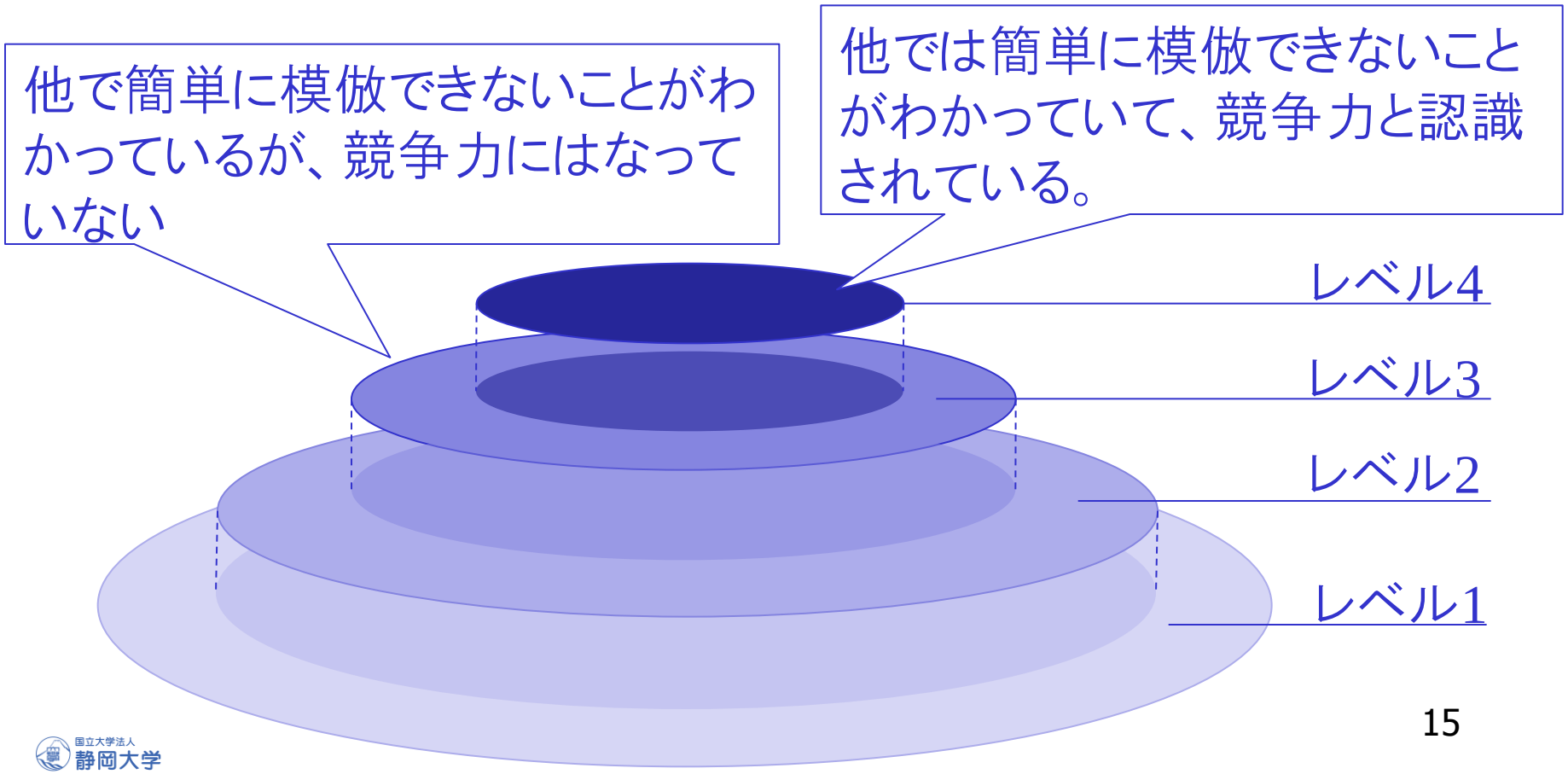
レベル4 – 競争力の源泉となっている(2／2)

- 自組織の強みが何かがわかり、方法がわかっている。
- 強みが顧客にとって高い価値を生み出している。



レベル3と4の違い

- 3 品質保証活動が特徴付けられている
- 4 品質保証活動が特徴付けられており、競争力の源泉となっている



レベル0からの脱却

- ふりかえりによって、活動を見直す。
  - 全体を見直す場を設定する。
  - 他での事例を示す。
  - 時間がかかることを覚悟する。

- レベルの移行
  - レベル0、1からは2への移行を目指す。
  - レベル2からの移行と比較して難易度が高い。
  - レベル2への移行とレベル2からの移行は種類が異なる。

レビューで誤字脱字・表記の誤りの検出を控える

- 試行手順
  - 1. レビュー1回目: いつものやり方でレビューする。
  - 2. 誤字脱字の検出が最終的なシステムの品質向上に役立たないことを確認し合意する。
  - 3. レビュー2回目: 誤字脱字を検出しないよう依頼する。
- レビュー対象
  - A4 4ページ(日本語で記述)の簡単な仕様書 × 2
    - 1回目: テニスコート予約システム
    - 2回目: 社内勉強会参加申込みシステム
- 被験者
  - 実務者32名
  - 1回目、2回目とも、1人で20分程度欠陥検出してもらう。

出典: 森崎修司「意味あるレビューに改善できる?」日経SYSTEMS 2011年3月号, pp. 50-55(2011)

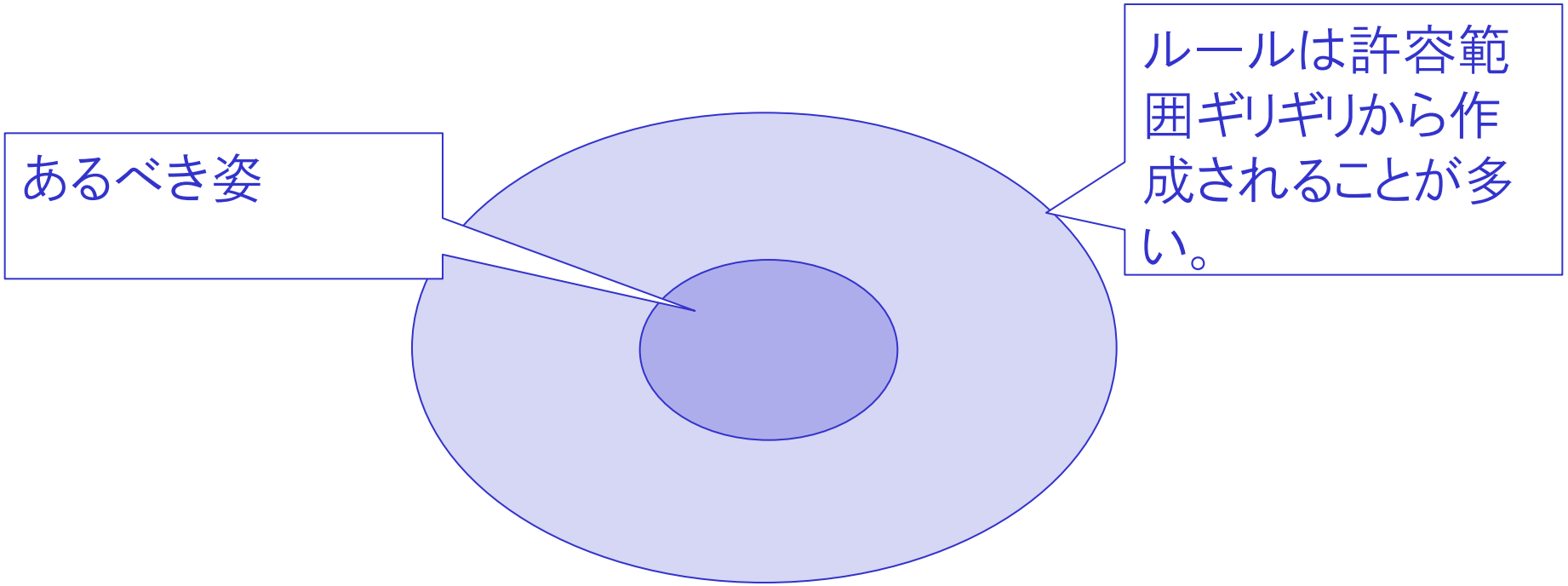
レビュー試行での合意の結果

- 誤字脱字、表記に関する指摘が大幅に減少した。
  - レビュー1: 24.4%
  - レビュー2: 6.7%
- 誤字脱字表記以外の指摘が増え、本質的な指摘が増えた。
  - 主要機能の記載漏れ
  - システム利用時の業務フローの問題
- レビュー指摘によって得られるメリット（修正コストの低減）を理解し、合意できた効果と考えられる。
  - 「レビューの効果は修正コストの低減である」という合意が得られた。
  - 誤字脱字、表記に関する指摘に使う時間を本質的な欠陥の検出に使えた。

出典： 森崎修司「意味あるレビューに改善できる？」日経SYSTEMS 2011年3月号, pp. 50-55（2011）

ルールの増加、強調はギリギリで守る人を増やす

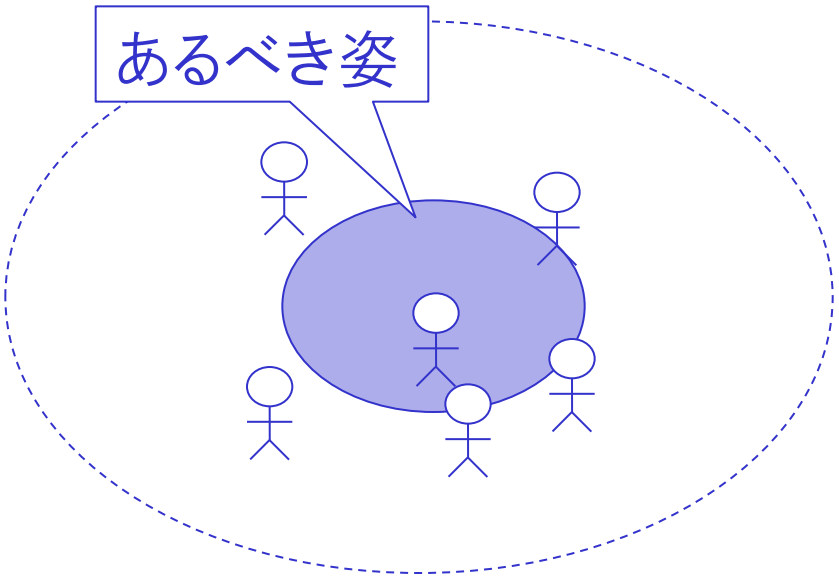
- ルールを遵守することを強調するとルールを守っていれば大丈夫という人を増やす。
- ルールを増やしていくとそれ以外は気にしなくてもよいという人を増やす。



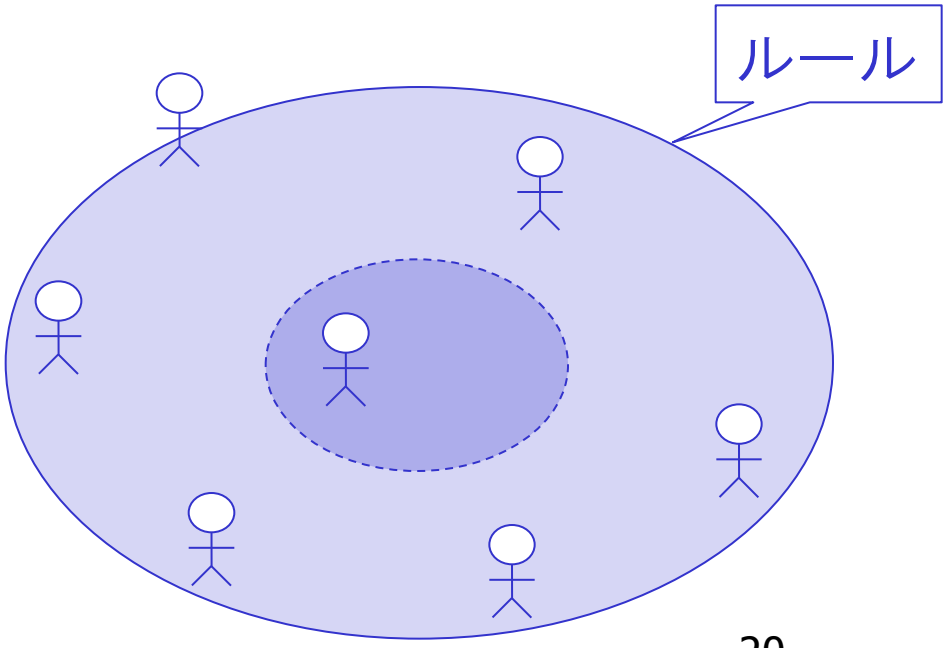
ルールの増加、強調はギリギリで守る人を増やす

- ルールを遵守することを強調するとルールを守っていれば大丈夫という人を増やす。
- ルールを増やしていくとそれ以外は気にしなくてもよいという人を増やす。

あるべき姿を示す場合



ルールを示す場合



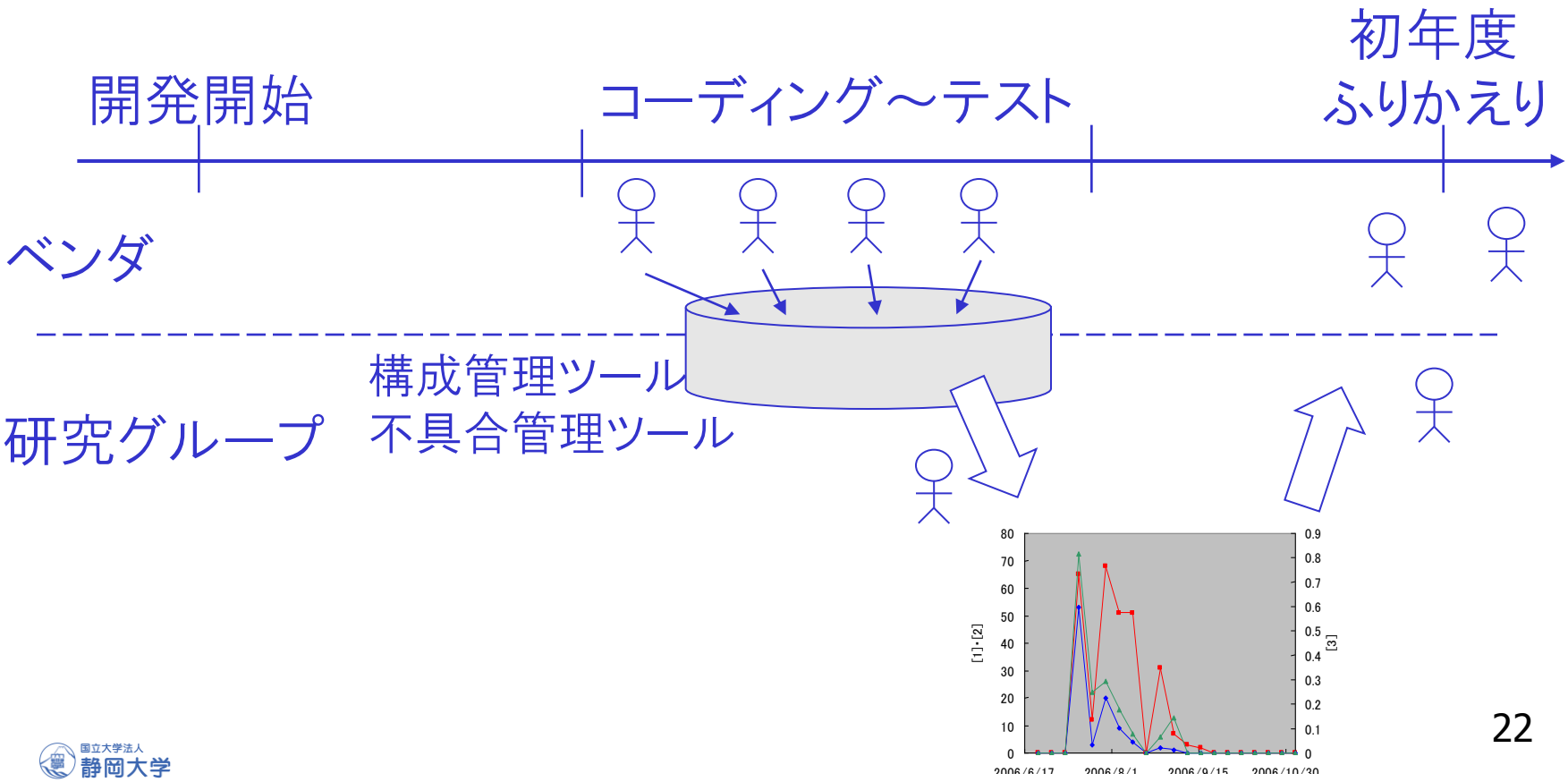
レベル1からレベル2へ：事例

- 対象プロジェクト
  - 経済産業省「先進社会基盤構築ソフトウェア開発事業」
  - 交通情報を中心としたセンサネットワークシステムのサーバサイドロジック
  - 自動車会社の要求をもとにNTTデータがマネジメント、デンソー、日本電気、パナソニック、日立製作所、富士通がベンダとして開発した。
- C/C++で330KLOC（流用込み）、約1年 × 2回
- 初年度と次年度で我々の研究グループがマネジメント企業とともに技法の活用や検証に取り組んだ。

参考： IPA SEC Books, ソフトウェアエンジニアリングの実践、エンピリカルソフトウェア工学の勧め
http://sec.ipa.go.jp/publish/index.html

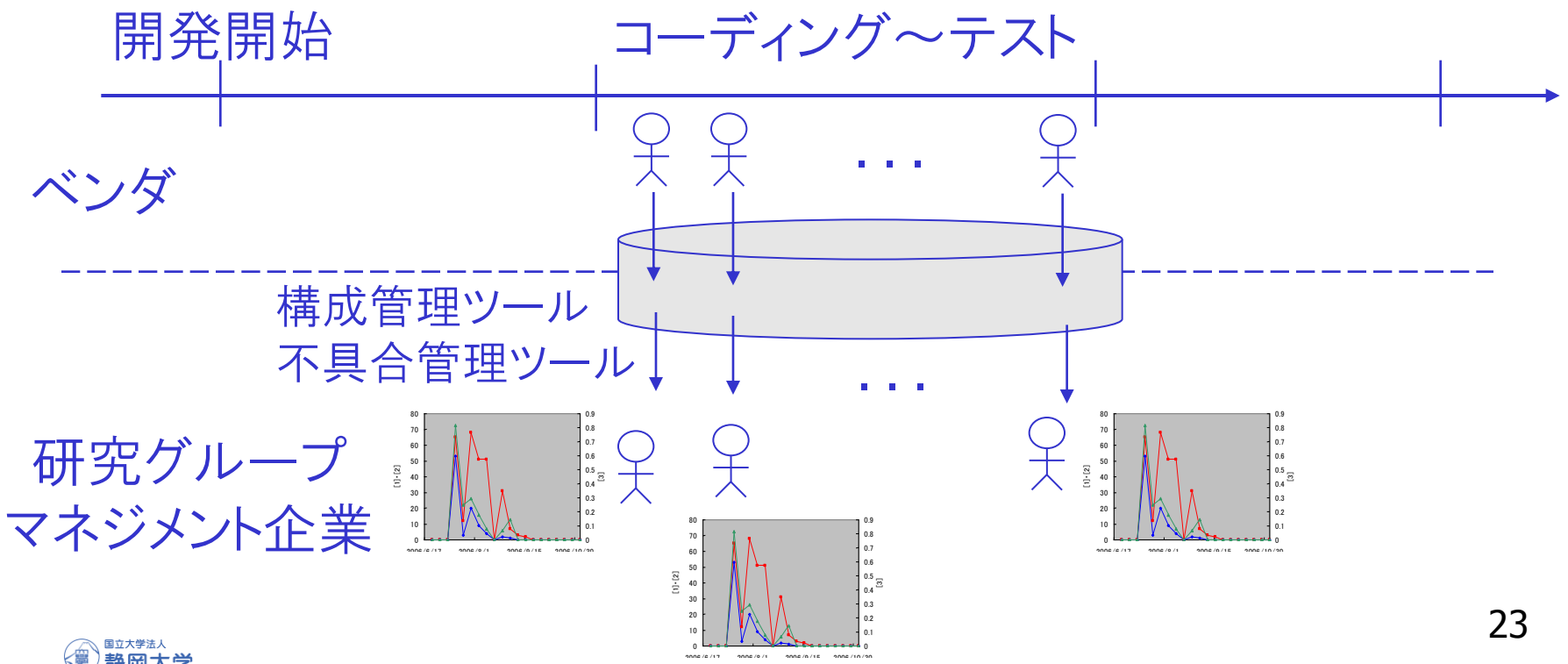
初年度

- 構成管理ツールからソースコードの規模遷移を時系列に可視化し、問題のありそうな部分をヒアリングした。
- 質問とフィードバックは事後に実施した。

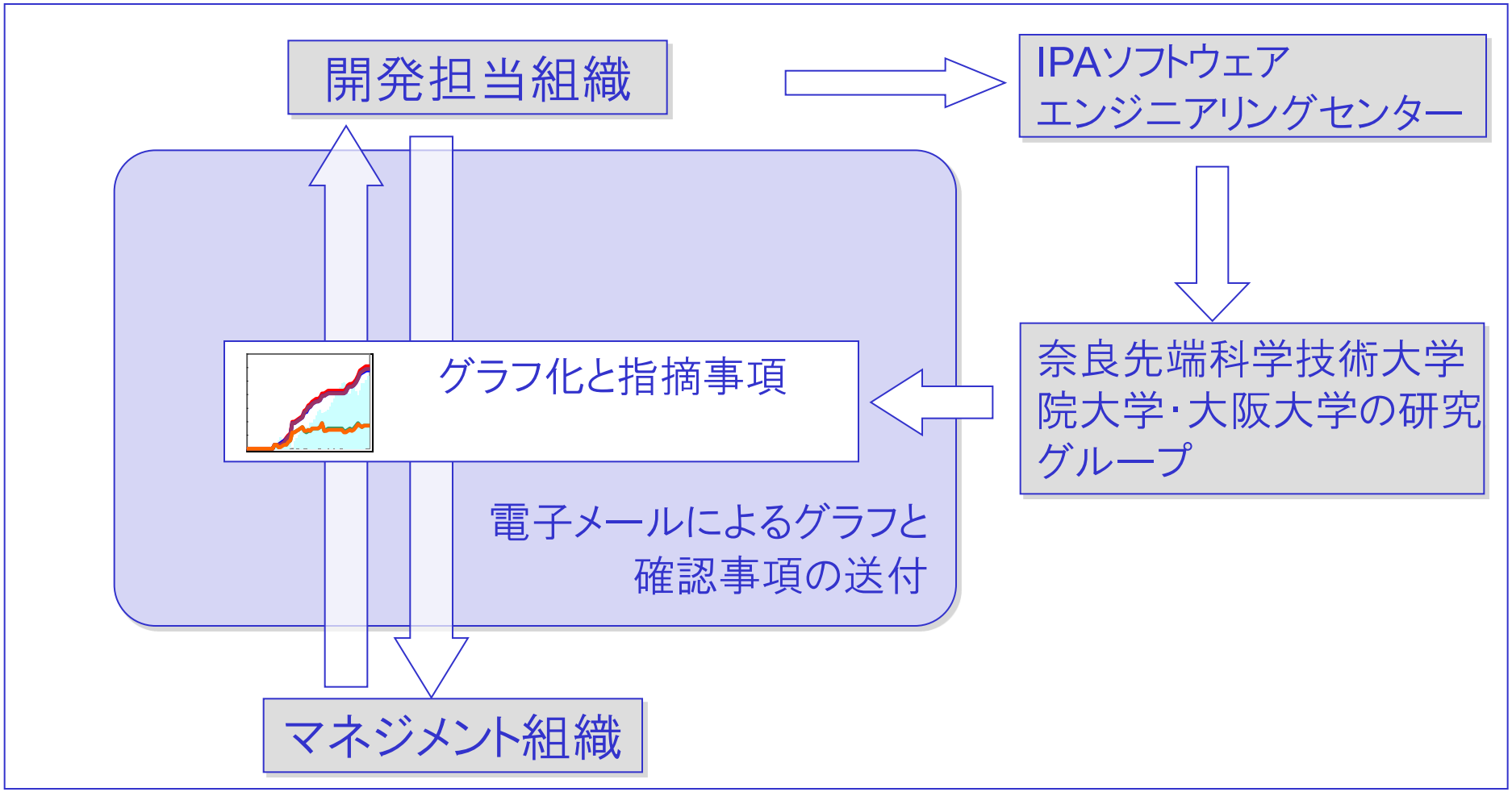


2年度目

- 週次ミーティングの前に、ソースコード規模遷移、不具合管理表のデータをもとに問題点の有無を確認することを合意
- 研究グループとマネジメント会社がデータの上で問題なしと認めた場合、週次ミーティングの一部を省略するルール



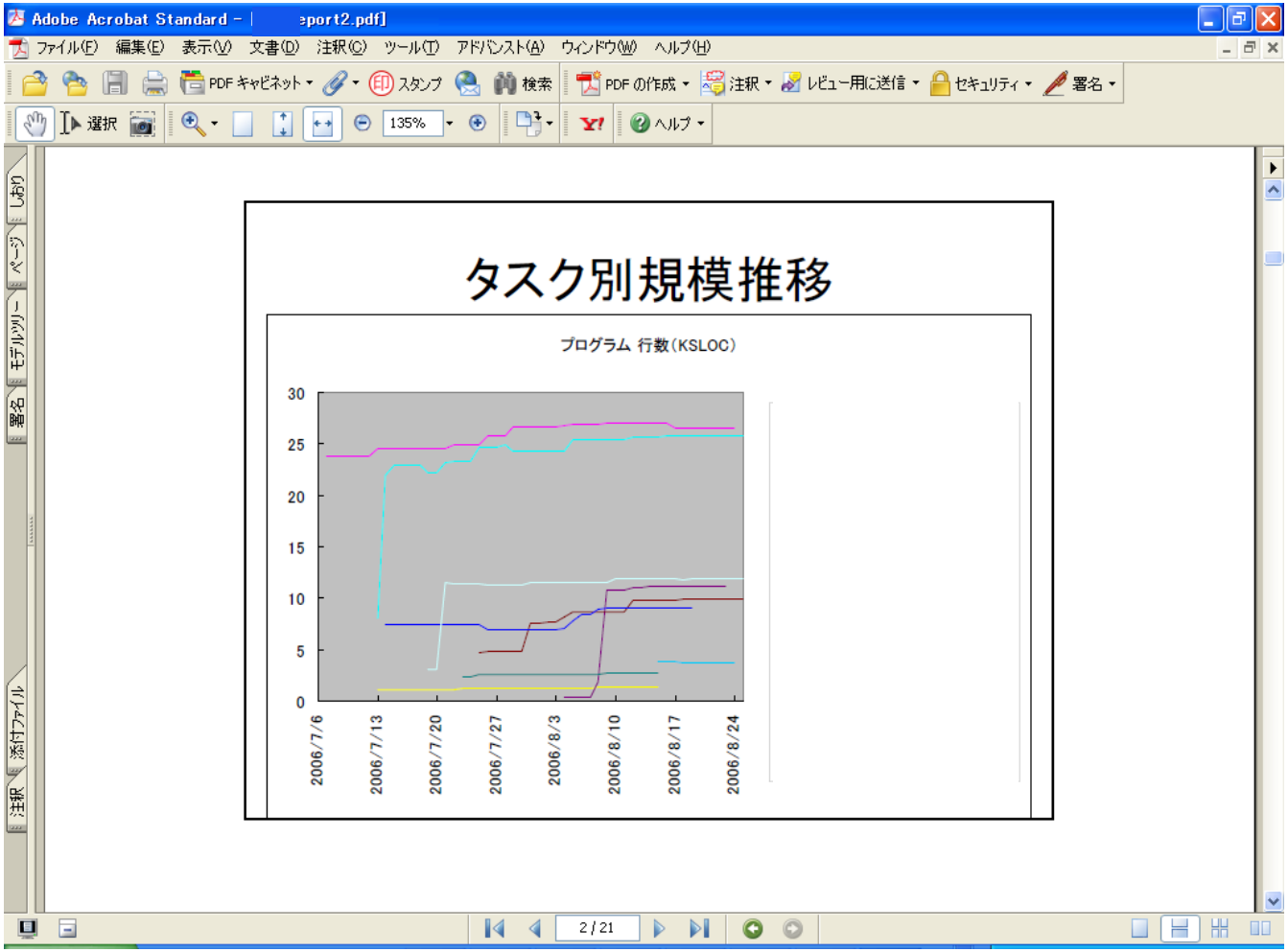
2年度目：電子メールによるフィードバック



前週までの実績をもとに毎週5社分

出典：　松村知子, 勝又敏次, 森崎修司, 玉田春昭, 大杉直樹, 門田暁人, 楠本真二, 松本健一, **自動データ収集・可視化ツールを用いたリアルタイムフィードバックシステムの構築と試行**, 奈良先端科学技術大学院大学テクニカルレポート TR2007001（2007/2）

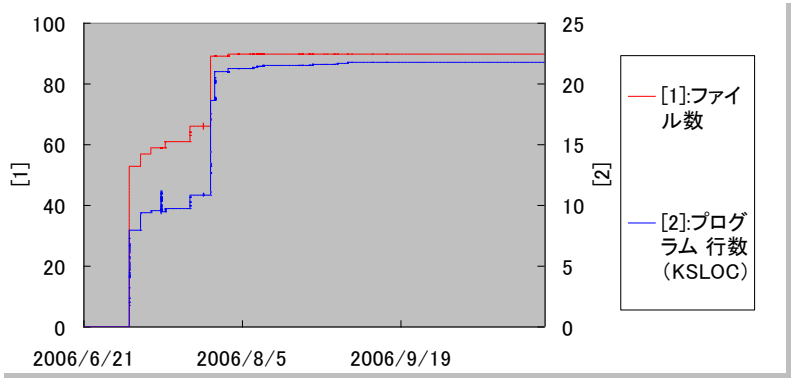
規模遷移のグラフ



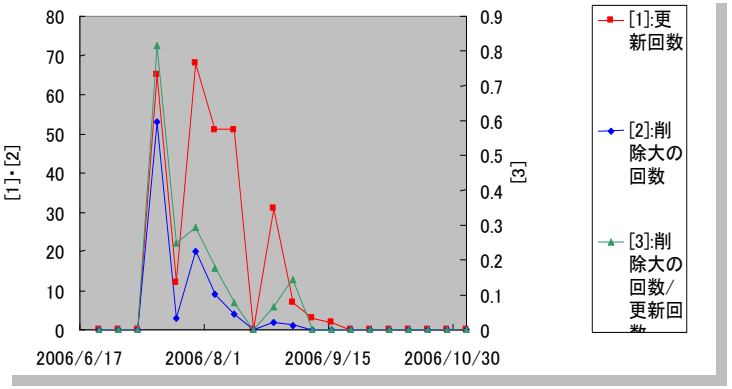
出典： 松村知子, 勝又敏次, 森崎修司, 玉田春昭, 大杉直樹, 門田暁人, 楠本真二, 松本健一, **自動データ収集・可視化ツールを用いたリアルタイムフィードバックシステムの構築と試行**, 奈良先端科学技術大学院大学テクニカルレポート TR2007001（2007/2）

プロダクト規模遷移を用いた開発リスク早期発見

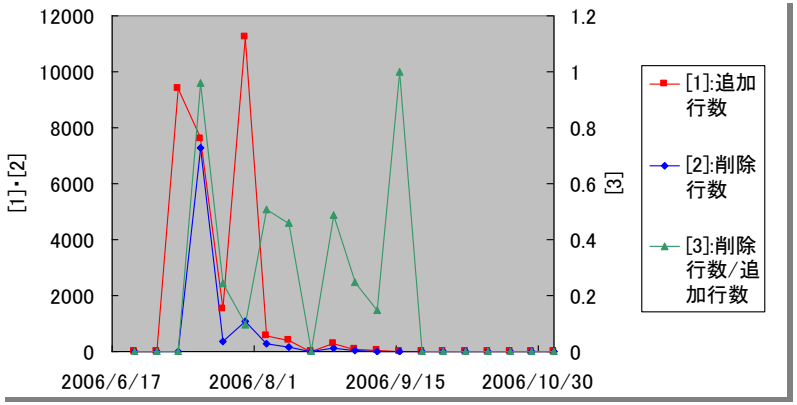
- 機能、サブシステム単位で規模遷移をモニタする。



1. ファイル数、行数



2. 更新頻度



3. 更新量

出典: 松村, 森崎, 勝又, 玉田, 吉田, 楠本, 松本 “問題の早期発見・改善を支援するインプロセスプロジェクト管理手法の実プロジェクトへの適用”, 電子情報通信学会論文誌Vol. J92-D, No. 11, pp. 1974～1986 (2009)

適用結果

- 対象ソフトウェアに関する知識が少ない、研究グループのメンバがプロダクト規模推移にもとづいて問題を指摘
- 問題が確認されない場合には進捗会議を短縮した。
- 当学のメンバも進捗会議に参加し、技法の洗練に努めた。

研究グループによるリスク指摘

	全指摘	異常指摘	問題の顕在化に寄与
件数	136	83	27
割合	100%	61%	20%

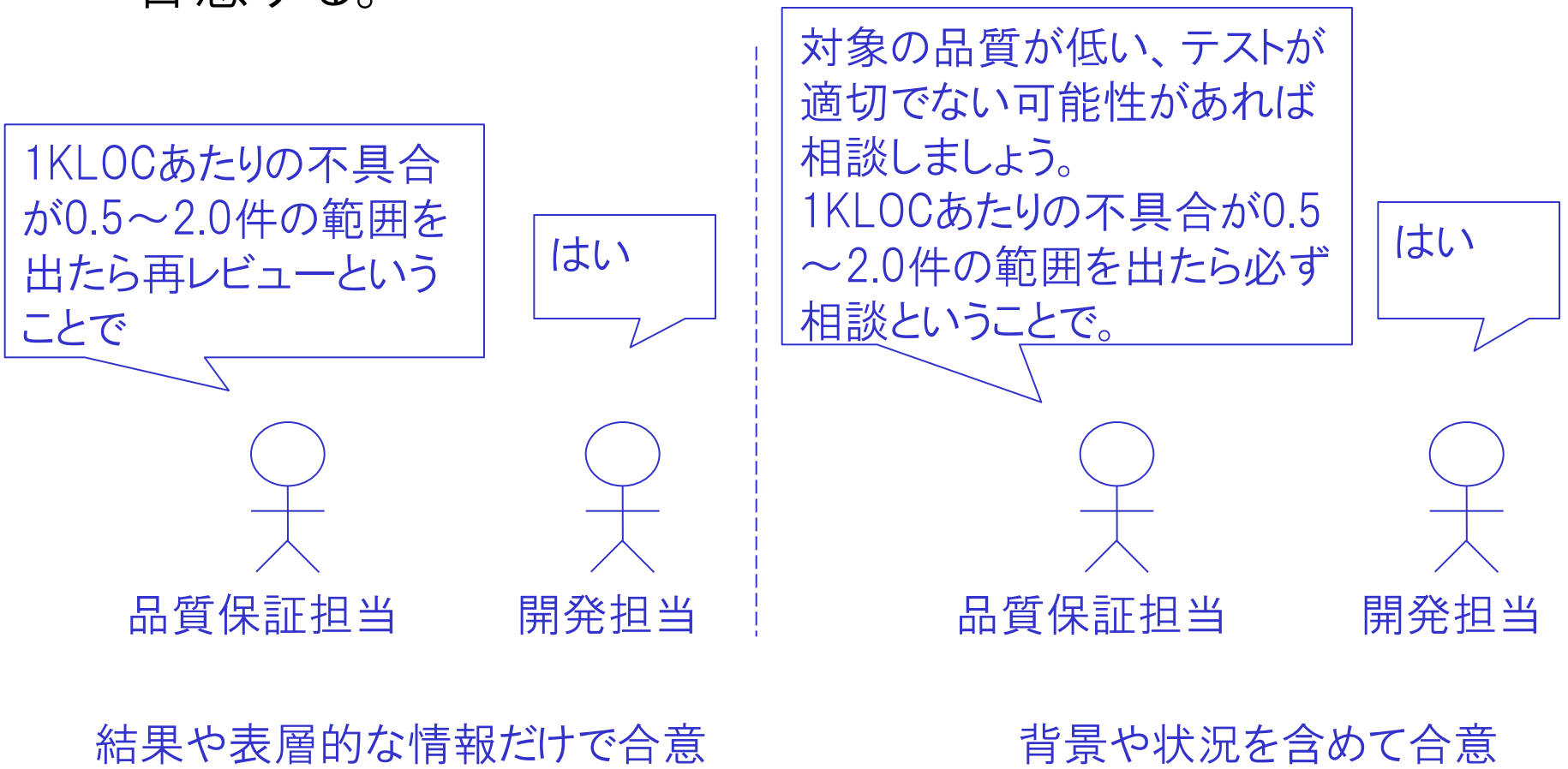
出典: 松村, 森崎, 勝又, 玉田, 吉田, 楠本, 松本 “問題の早期発見・改善を支援するインプロセスプロジェクト管理手法の実プロジェクトへの適用”, 電子情報通信学会論文誌Vol. J92-D, No. 11, pp. 1974～1986 (2009)

参考になる議論：DevOps

- Webを通じてサービスを提供する技術者の間で問題とされ議論されている。
  - Dev: Development (開発)
    新しい機能をリリースしたい。
  - Ops: Operations (運用): QAを含んでいることもある。
    安定してサービスを提供したい。
- 開発と運用の間で相反する意図や言い争いを減らし、ユーザに価値を届けるために必要な仕組みやマインド
  - 活動の透明性を高める。
  - 頻繁なリリースによって抵抗を小さくする。
  - ツールを活用する。

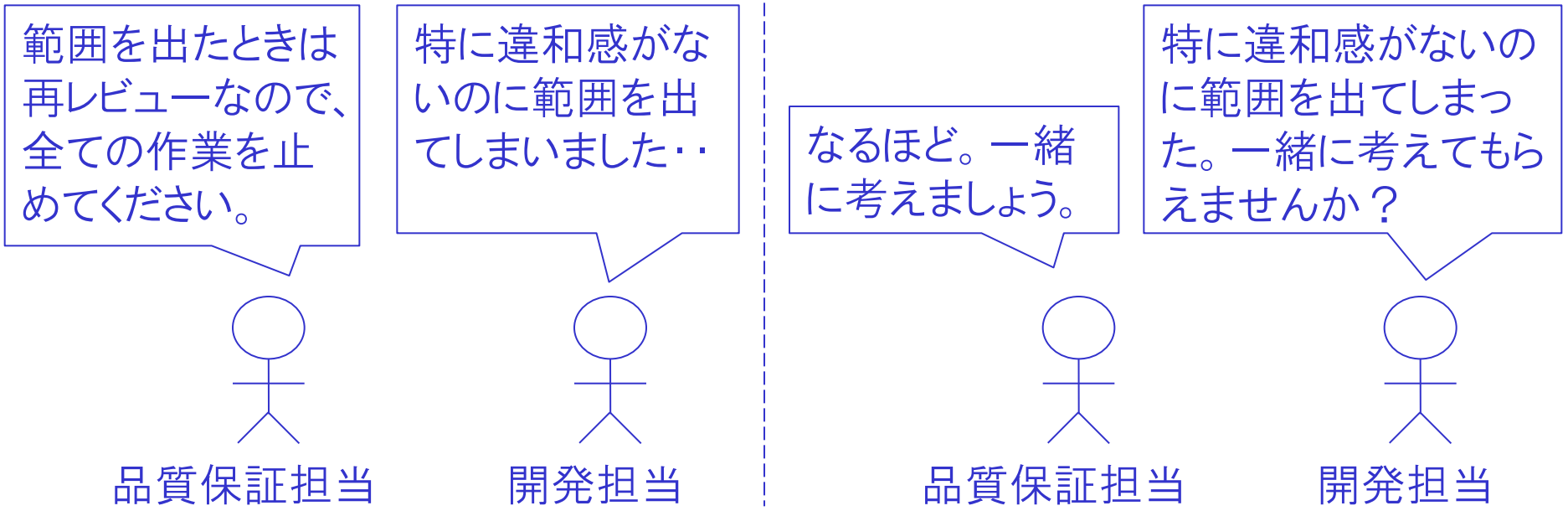
背景・状況を合意する

- 結果や表層的な情報だけではなく、背景や状況を含めて合意する。



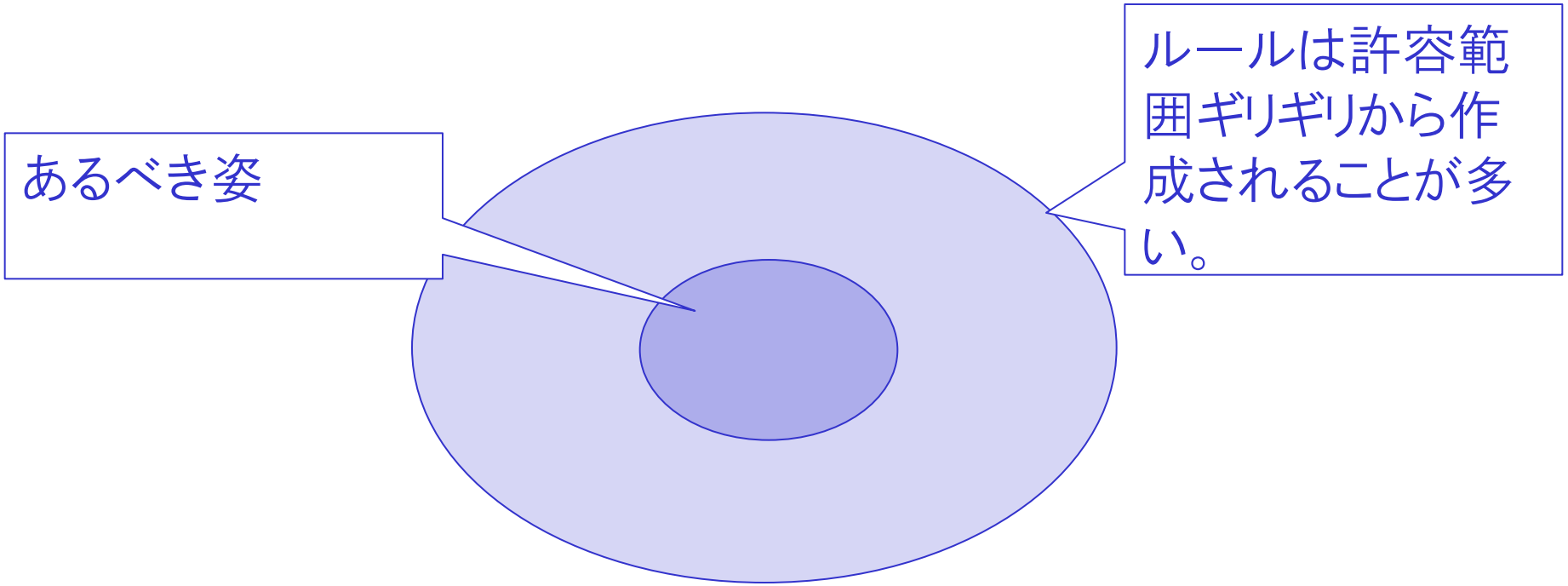
背景･状況を合意する

- 結果や表層的な情報だけではなく、背景や状況を含めて合意する。



ルールの増加、強調はギリギリで守る人を増やす（再掲）

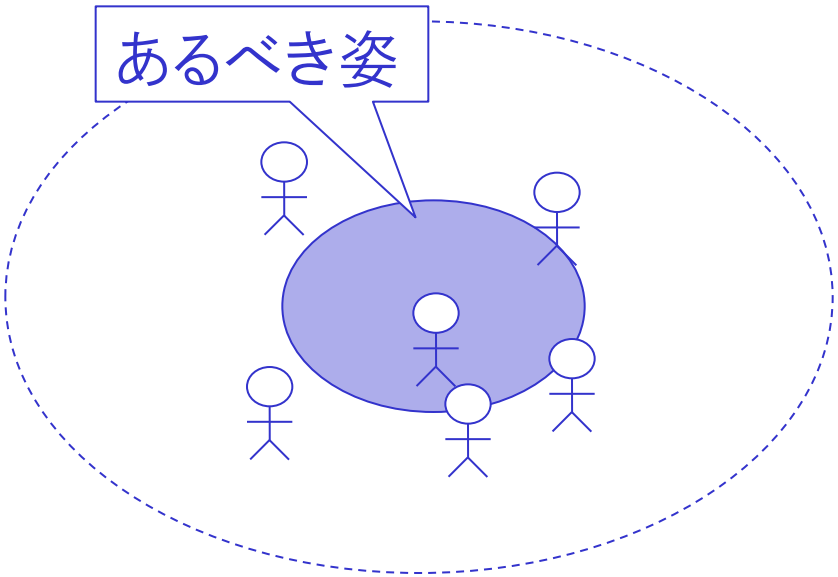
- ルールを遵守することを強調するとルールを守っていれば大丈夫という人を増やす。
- ルールを増やしていくとそれ以外は気にしなくてもよいという人を増やす。



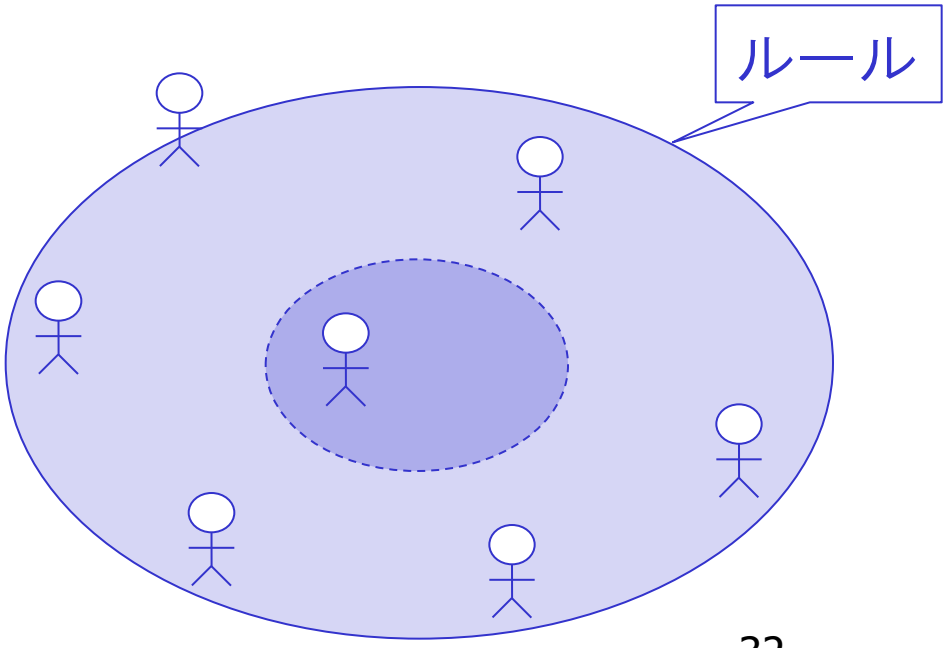
ルールの増加、強調はギリギリで守る人を増やす（再掲）

- ルールを遵守することを強調するとルールを守っていれば大丈夫という人を増やす。
- ルールを増やしていくとそれ以外は気にしなくてもよいという人を増やす。

あるべき姿を示す場合

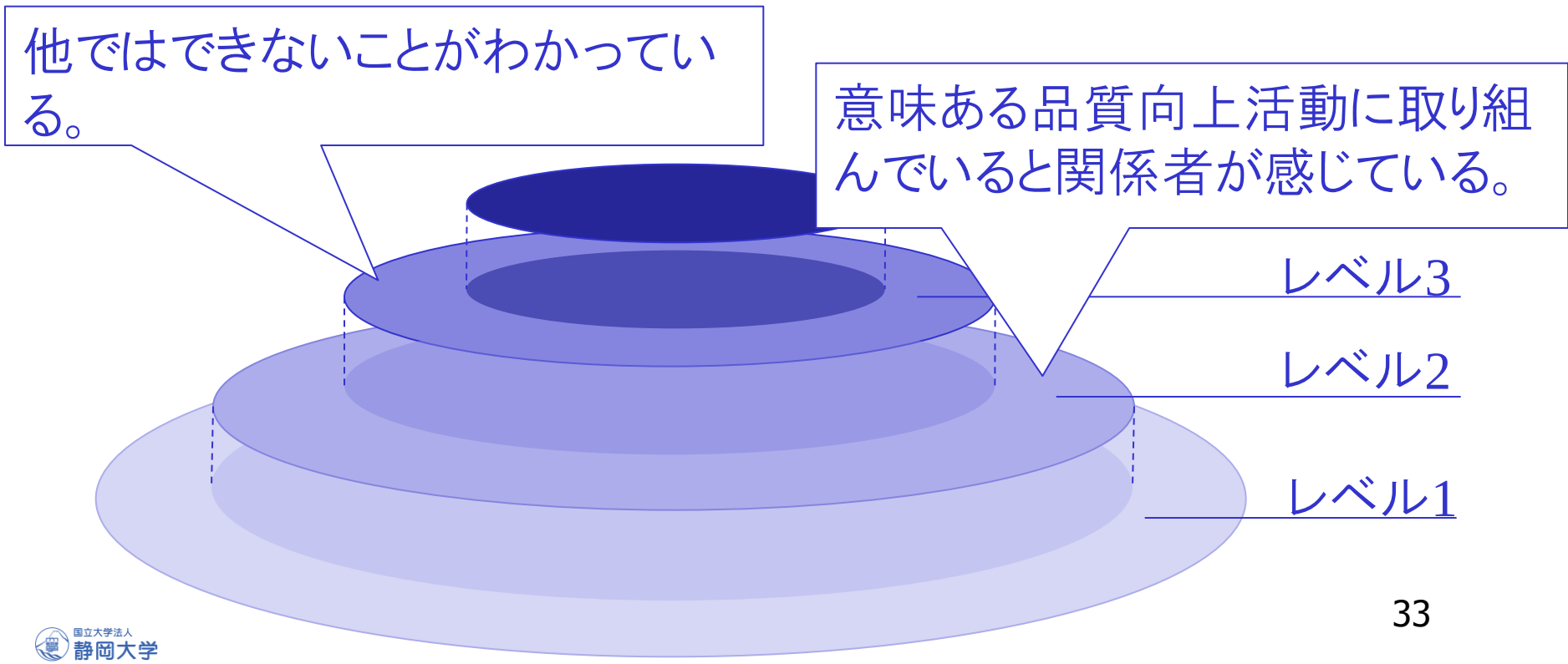


ルールを示す場合



レベル2と3の違い（再掲）

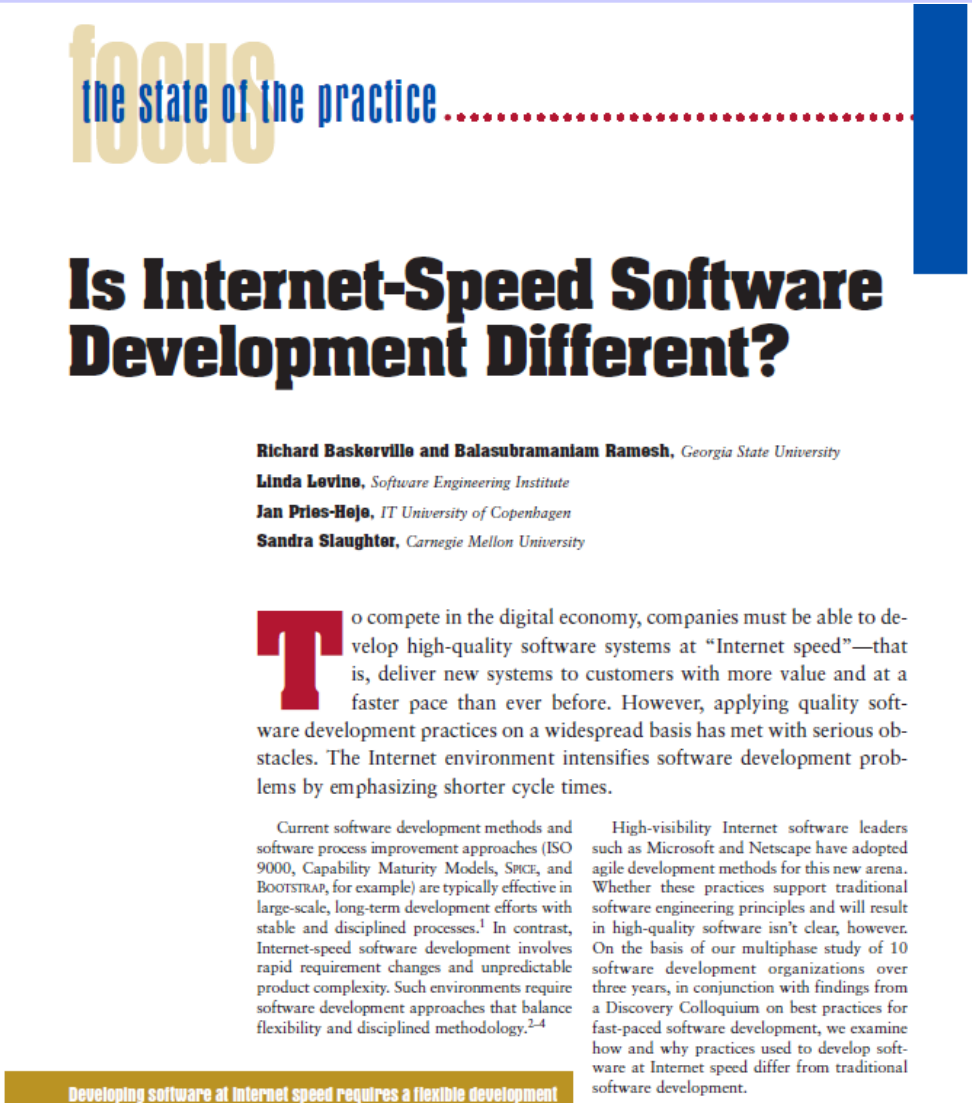
- 2 関係者が合意した上で品質保証活動ができている
- 3 品質保証活動が特徴付けられている



レベル2からレベル3へ

- うまくいっている理由を深掘りする（成功要因の検討）。
  - 期待通り機能している品質向上活動の限界や制約を知る。
  - コンテキストを明確にする。
- うまくいっている理由をもとに自組織の特徴と関連づける。
  例）
  - 人事評価にアシストポイントが加味されている → レビューに積極的
  - となりの部門であっても口出しして、改善することが阻まれない。→ 問題の早期発見
  - 失敗事例を共有することが善とされている。→ ノウハウ共有
- 既存の技法、技術を再構成しなければならない場合もある。
  - 新規に習得する必要は必ずしもない。
  - 既存の技法、技術のよい部分は残す。

「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」- コンテキスト分析例

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 [illegible] logically cohesive releases.

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 problems that can occur in long-lived, tightly coupled, and complex software.

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, I. Levine and S. Slaughter: Is Internet-Speed Software Development Different?, IEEE Software, vol. 20, no. 16, p70-77(2003)

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。

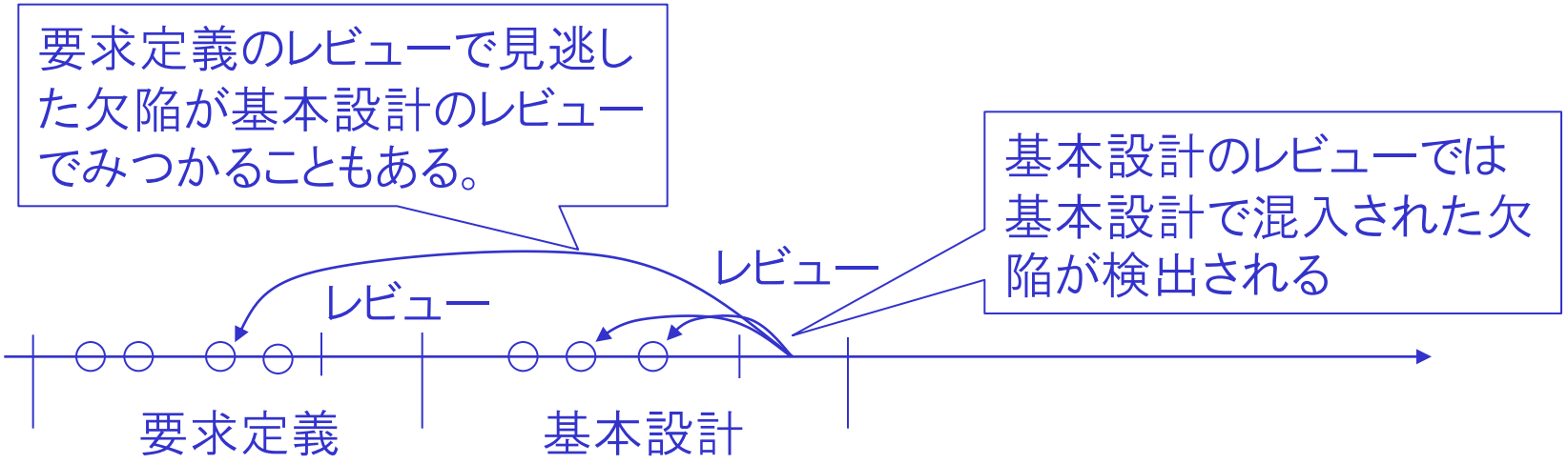
出典: R. Baskerville, B. Ramesh, L. Levine and S. Slaughter: Is Internet-Speed Software Development Different?, IEEE Software, vol. 20, no. 16, p70-77(2003)

レベル2からレベル3へ

- 少し離れた視点で考える必要がある。
- コンテキストの抽出は考えればわかる場合もあるが、他での事例や状況を知らなければ難しいこともある。
- オープンなシンポジウム、カンファレンス、記事等で・・
  - 自分達のほうが優れている点はどこか？
  - 自分達でもできることは何か？
  - 自分達ではできないことは何か？

見逃し率

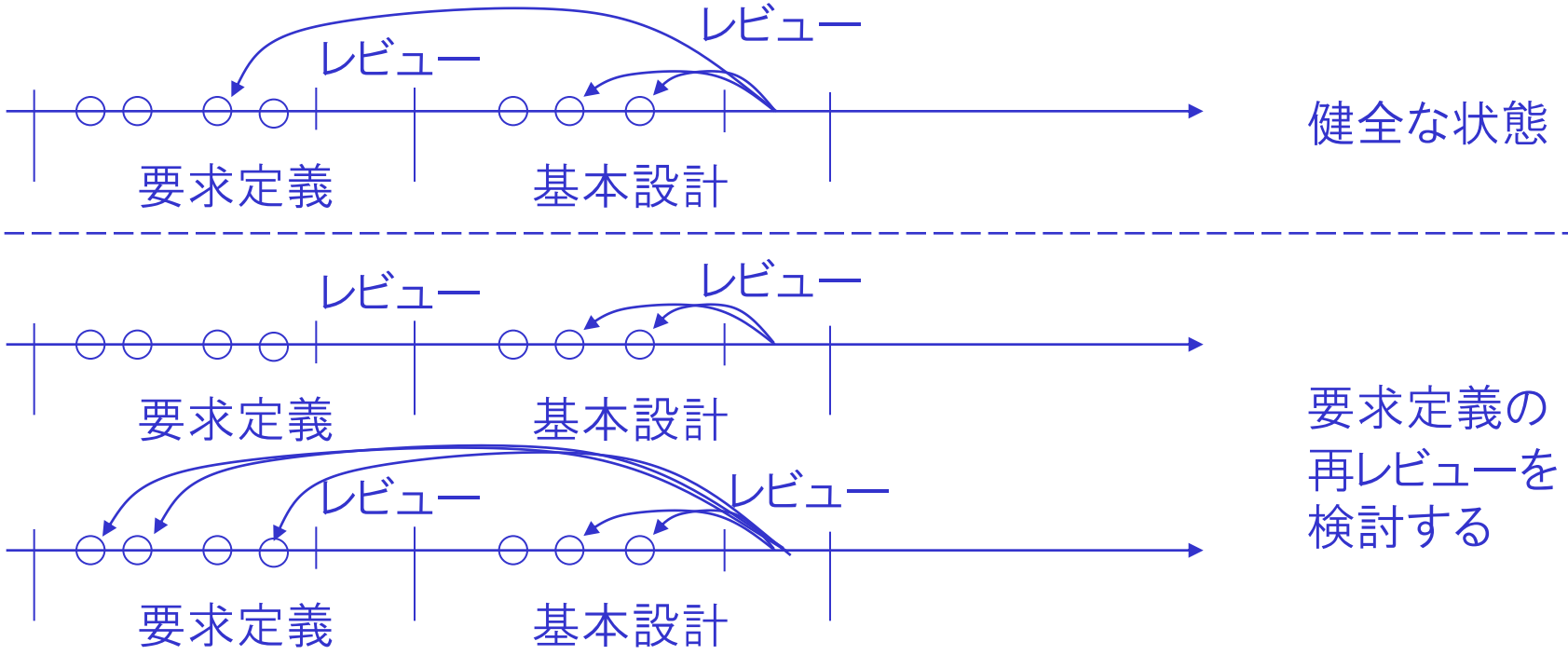
- レビューにおいて1つ前の工程の見逃し欠陥に上限･下限を設定し、再レビューの契機やレビュー評価に使う。
  - 見逃し欠陥がゼロということはないという前提に立つ。
  - 1工程前の見逃し欠陥を報告しやすい土壌を作る。
  - 最終成果物の品質向上を強く意識する。



出典:清田 辰巳、上流工程における発注者視点からの品質向上への取り組み (特集 ソフトウェアレビュー/ソフトウェアインスペクションと欠陥予防の現在) 、情報処理学会　学会誌 Vol. 50, No. 5, p. 400-404　(2009)

見逃し率

- レビューにおいて1つ前の工程の見逃し欠陥に上限･下限を設定し、再レビューの契機やレビュー評価に使う。
  - 見逃し欠陥がゼロということはないという前提に立つ。
  - 1工程前の見逃し欠陥を報告しやすい土壌を作る。
  - 最終成果物の品質向上を強く意識する。



出典:清田 辰巳、上流工程における発注者視点からの品質向上への取り組み (特集 ソフトウェアレビュー/ソフトウェアインスペクションと欠陥予防の現在) 、情報処理学会　学会誌 Vol. 50, No. 5, p. 400-404

FixCacheによる不具合予測の導入

- 最先端の研究成果を実際の開発に取り入れる。
- FixCache*[1]: 過去に修正された不具合と同じような不具合が再発する。
  - 7つのオープンソースプロジェクトの変更履歴約200,000件で実証
  - 「メモリキャッシュ」のアルゴリズムを用いて不具合の可能性の高いソースコードファイルを選出
  - 選出した全体のソースコードファイルの1割で73～95%の精度で不具合を予測できた。
- Googleでのソースコード更新履歴に基づく不具合予測に応用されている。*[2]

*1: Kim S., Zimmermann T., James E. W., Zeller A., “Predicting Faults from Cached History” In Proceedings of the 29th international conference on Software Engineering , p. 489-498.

*2: Lewis C., Ou R. “Bug Prediction at Google” http://google-engtools.blogspot.jp/2011/12/bug-prediction-at-google.html

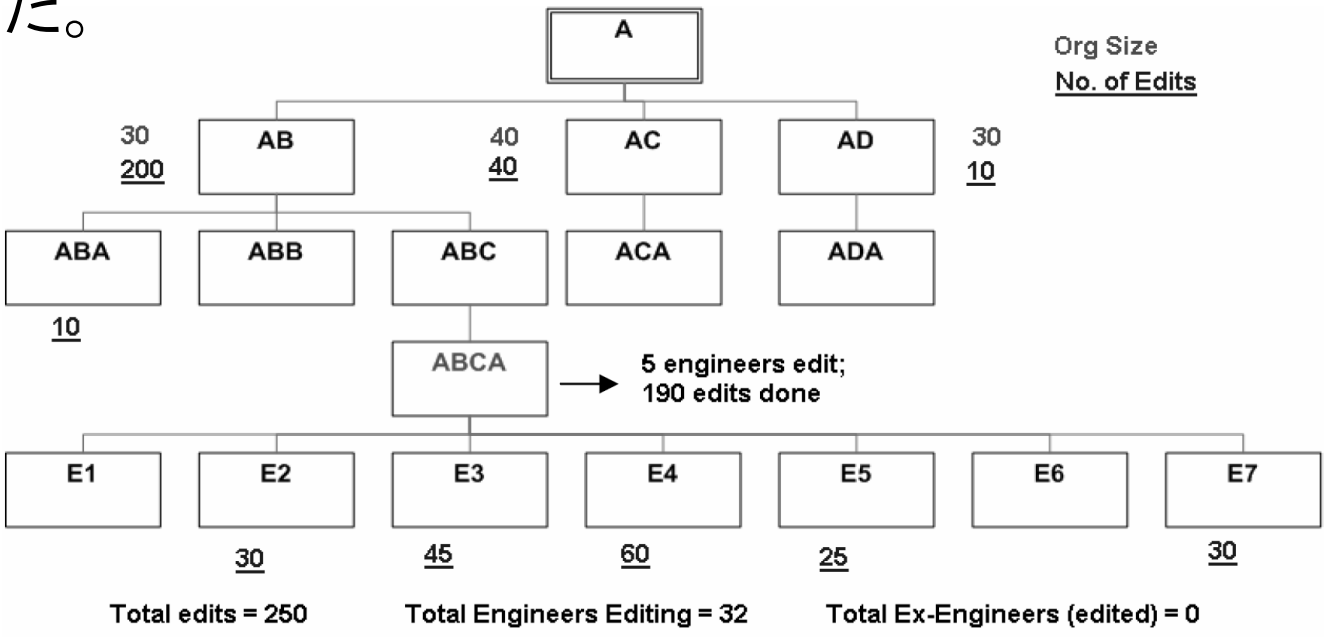
世界規模の分散開発

- コンウェイの法則
  「製品の設計は組織のコミュニケーションの構造を反映したものになる」 M.E. Conway, “How Do Committees Invent?” Datamation, vol. 14, no. 4, pp. 28-31(1968)
  http://www.melconway.com/research/committees.html

- 商用ソフトウェアの分析によるコンウェイの法則の裏付け
  - メンバのコミュニケーションが密接に取りにくいモジュールほど品質が低い傾向にある。

商用ソフトウェアでの裏付け

- ソフトウェアモジュールの不具合予測として、組織の構造を使う。
- 編集の75%を超える組織の距離と残存不具合数の関係を調べた。



出典: N. Nagappan, B. Murphy, and Victor Basili, “The influence of organizational structure on software quality: an empirical case study,” In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, pp. 521-530. http://research.microsoft.com/apps/pubs/default.aspx?id=70535

商用ソフトウェアでの裏付け（イメージ）

- Windows Vistaのモジュールで試した。
- プロダクト、プロセスメトリクスよりも組織構造のメトリクスのほうが品質の予測精度が高かった。

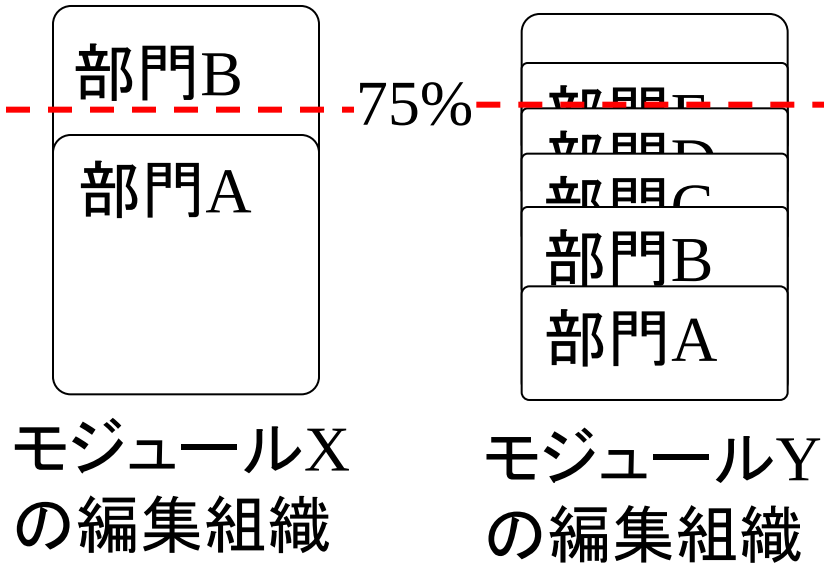


Table 4: Overall model accuracy using different software measures

Model	Precision	Recall
Organizational Structure	86.2%	84.0%
Code Churn	78.6%	79.9%
Code Complexity	79.3%	66.0%
Dependencies	74.4%	69.9%
Code Coverage	83.8%	54.4%
Pre-Release Bugs	73.8%	62.9%

出典: N. Nagappan, B. Murphy, and Victor Basili, “The influence of organizational structure on software quality: an empirical case study,” In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, pp. 521-530. http://research.microsoft.com/apps/pubs/default.aspx?id=70535

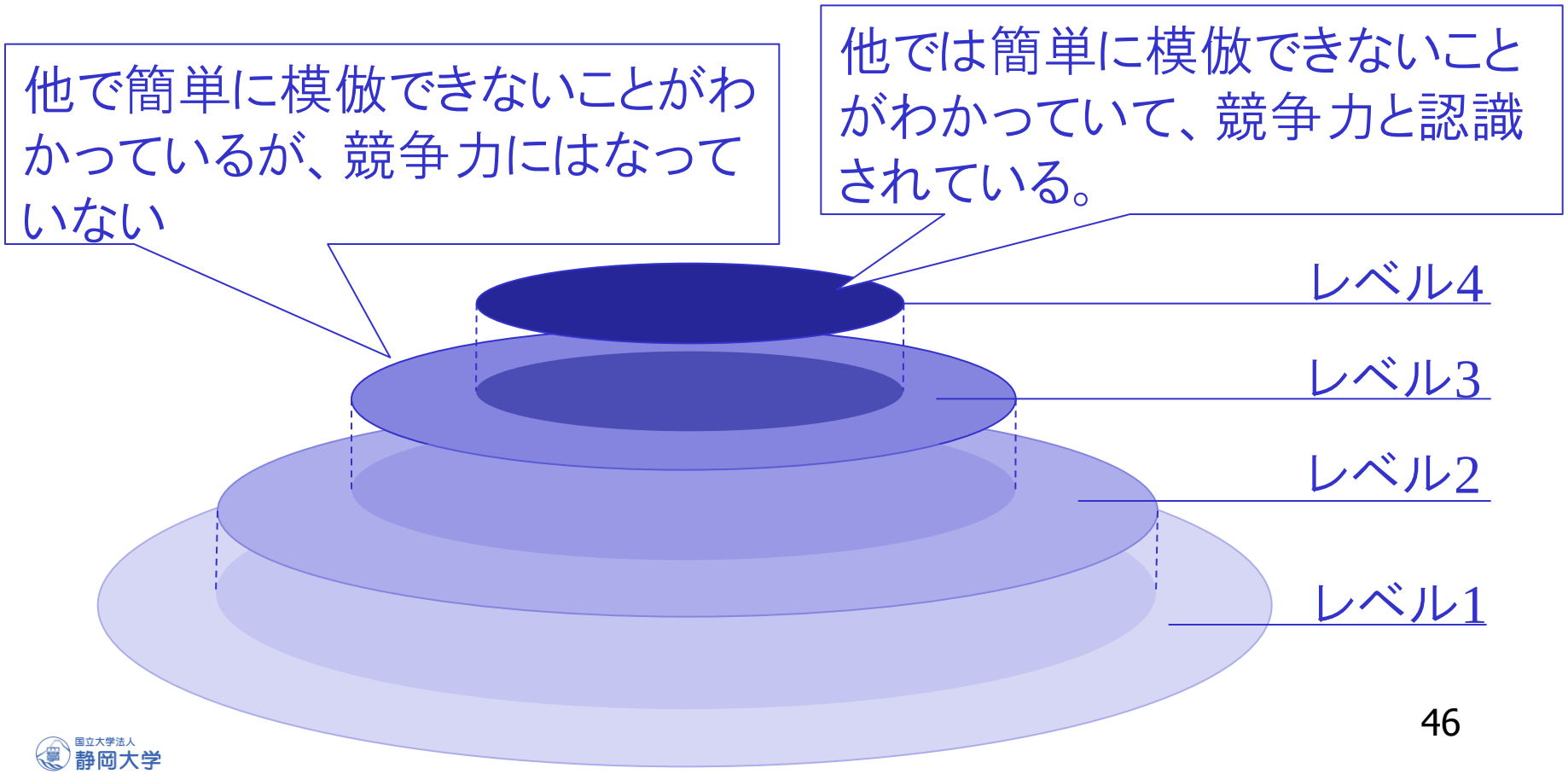
フィードバックの受け方

- Software evolutionの法則（リーマンの法則）
  - 使われるシステムは変わり続ける。
  - 対策をしない限り、継続して変更するとソフトウェアがだんだん複雑になる。
  - システムの進化はフィードバックの内容やプロセスによって決まる。
- ソフトウェアのライフサイクルを実データをもとに分類した論文*による（1980）
  - IBM OS/360の最初の25リリース分を分析した結果
  - プログラムのライフサイクルを定義し、保守を定義
  - 観察された現象を法則としてまとめている。

* M. Lehman, “Programs, Life Cycles, and Laws of Software Evolution” IEEE, Vol. 68, No. 9, pp. 1060-1076 (1980)

レベル3と4の違い（再掲）

- 3 品質保証活動が特徴付けられている
- 4 品質保証活動が特徴付けられており、競争力の源泉となっている



レベル3からレベル4へ

- 競争力を分析する。
  - レベル3の段階でレベル4はある程度視野に入るときもある。
  - 「あれ？」と思うところからはじまる。
  - 受託であれば失注の原因からわかる場合もある。
  - コミュニティ内でのあまり関係のない会話から気づくこともある。
- 品質向上活動のバックグラウンドを考える。
  - 体制によるもの
  - 文化（自然な考え方）によるもの
- 以降では特徴的な事例を紹介するので、ご自身のソフトウェアや組織であればどうか考えるきっかけにしていただきたい。

事例1）ビジネスに大きく影響する品質を早期に作り込む

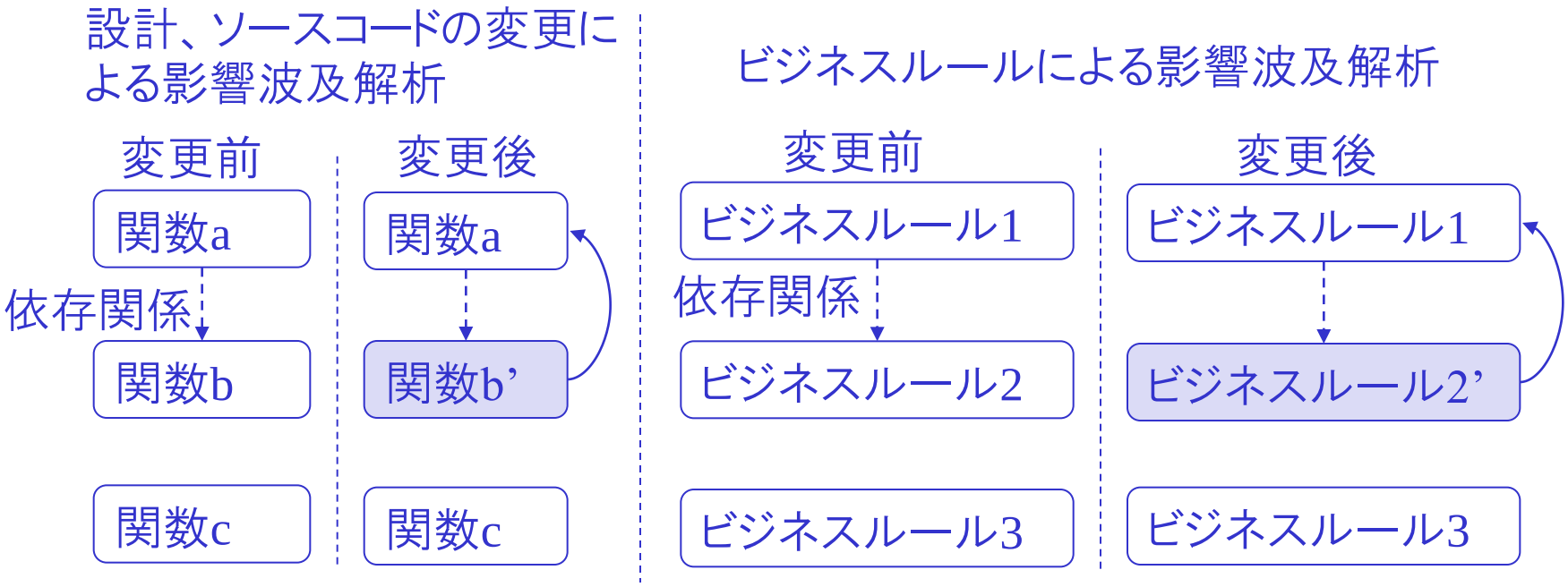
- 株式売買システムのスループットを高めるために開発早期からスループットを意識して作り込む。
  - 条件を事前に設定しておき、条件を満たせば売買する「アルゴリズム取引」ではトレーダがスループットを重視する。
  - 多くのトレーダが集まれば株式売買システムがより多くの利益を生み出す。

- 要求定義時点から過去の株式売買のパターン（オークションパターン）をもとに詳細化する。
  - 目標スループットの達成
  - 設計、先行検証、テスト

出典: 宇治 浩明、三澤 猛、“株式売買システムarrowhead開発事例”, ソフトウェア品質シンポジウム2011, http://www.juse.or.jp/software/366/attachs/sqip2011_A3.pdf (2011)

事例2）ビジネスルール・ビジネスプロセスから影響分析

- 通常、設計、ソースコード変更箇所からレビュー、テスト箇所を特定する（影響波及解析）。
- ビジネスルール・ビジネスプロセスのみで影響波及解析ができることを示している。



出典: Aryani A., Peake I., Hamilton M., “Domain-Based Change Propagation Analysis: An Enterprise System Case Study”, Proceedings of the 2010 IEEE International Conference on Software Maintenance, p. 1-9 (2010)
Slideshare: http://www.slideshare.net/amiraryani/domainbased-change-propagation-analysis-an-enterprise-case-study-5205028

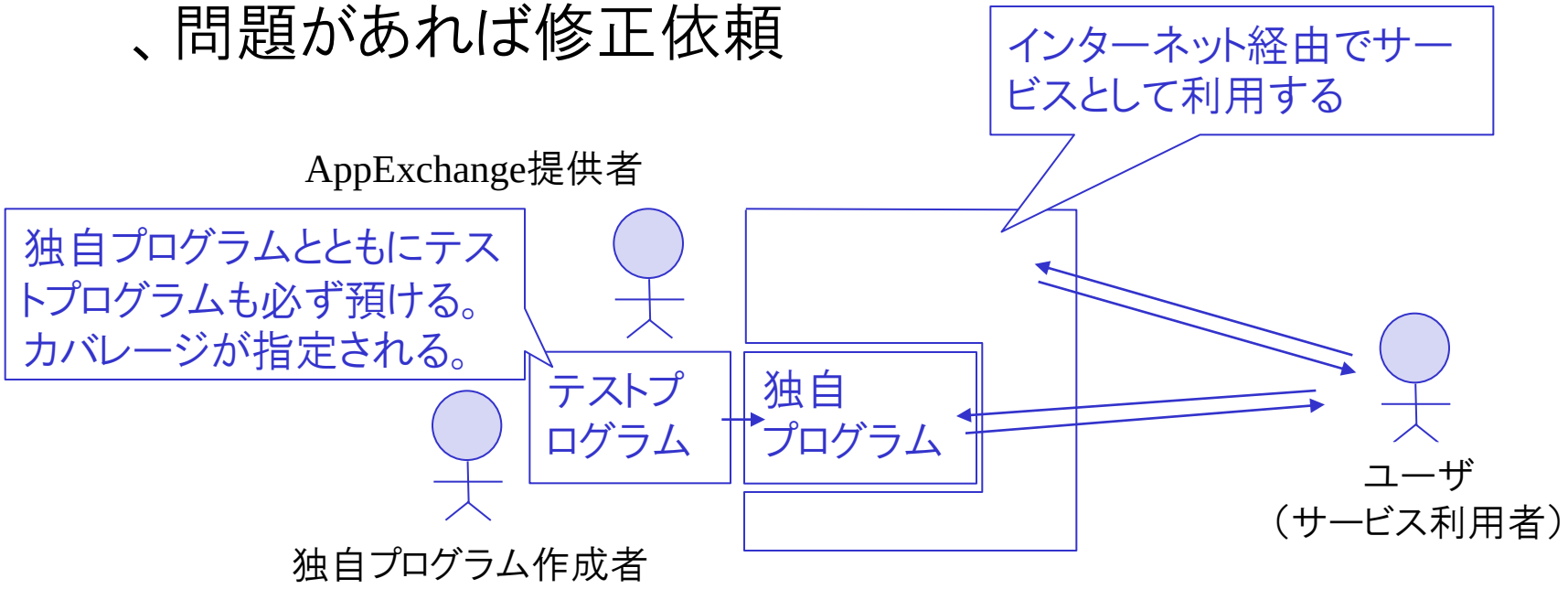
事例3）保守・拡張開発のテスト

- 顧客が作成した多数のソフトウェアを低コストで運用する。
  - 顧客の言い分を聞くと多様なOSやミドルウェアを運用しなければならない。
  - 同一のOSやミドルウェアに統一すると顧客が集められない。
- 顧客のソフトウェアを運用する際に顧客が作成したテストコードも同時に預かる。
  - OSやミドルウェアは単一バージョンとする。
  - バージョンアップ時にテストコードを実行し、問題があれば顧客に知らせる。
  - テストコードのカバレージを指定し、カバレージに満たない場合は預からない。

* Fisher S. “The Architecture of the Apex Platform, salesforce.com's Platform for Building On-Demand Applications”, Keynote Speech International Conference on Software Engineering 2007

事例3）AppExchangeでの方針

- 数千を超える顧客に対して同一バージョンを提供する。（個別対応はしない）
- 顧客のアプリケーションは一定カバレージ（75%）以上のテストコードとともにホスティングする。
- OS、ミドルウェア等のアップデート時には事前テストを実施し、問題があれば修正依頼



* Fisher S. “The Architecture of the Apex Platform, salesforce.com's Platform for Building On-Demand Applications”, Keynote Speech International Conference on Software Engineering 2007

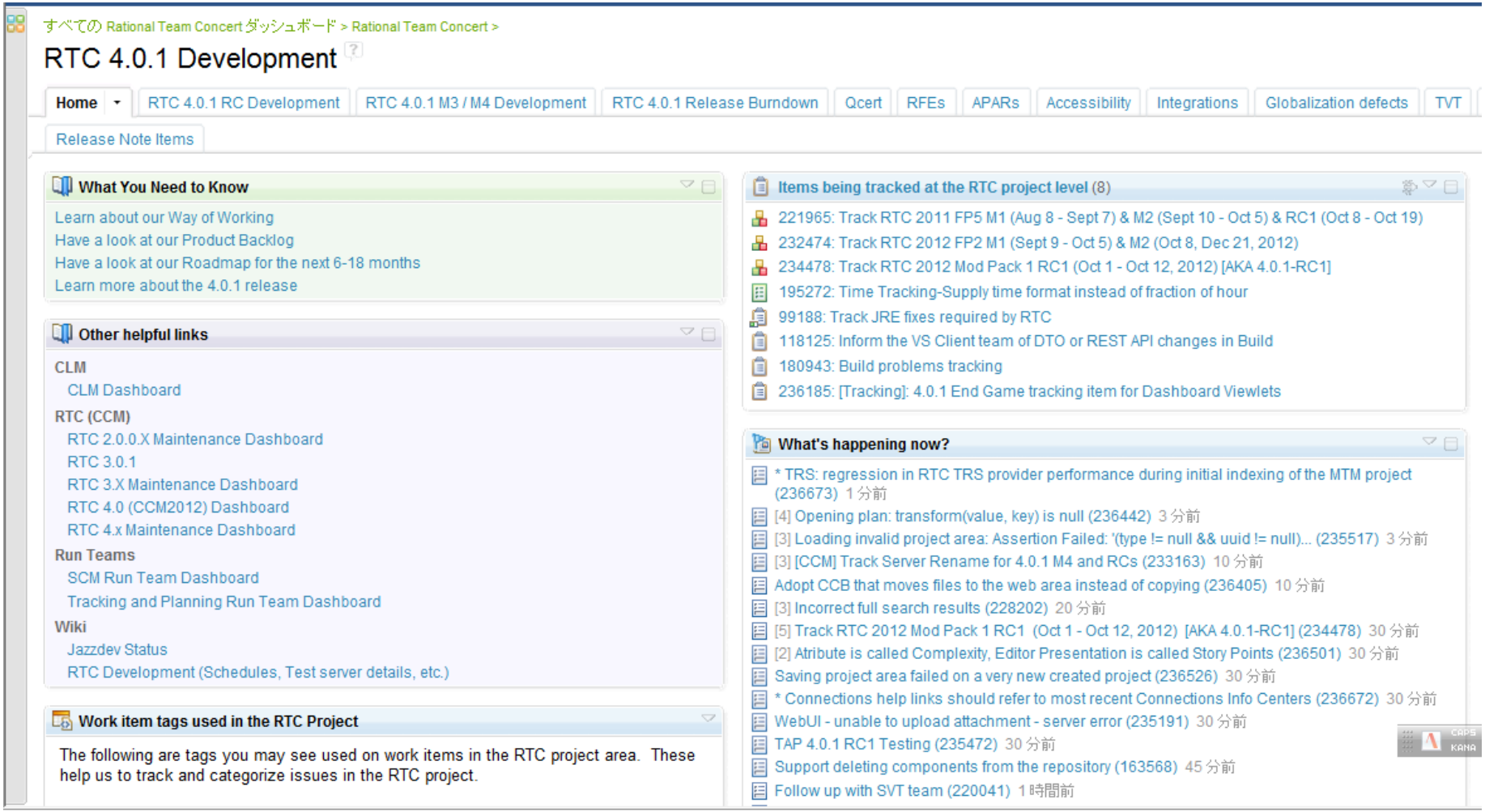
事例4）Twitter ダークモード

- 機能を細かく分解し個別に実行を制御できるアーキテクチャを持つ。
- 部分的に機能を使えない状態にして運用することをダークモードと呼ぶ。
  - 頻繁に機能をリリースする（continuous delivery）。
  - 問題のある機能のみ一時的に実行できなくする。
- 機能に対してユーザが直接対価を支払わない。

John Adams, “In the Belly of the Whale: Operations at Twitter” , Velocity 2010(オライリーのカンファレンス), http://www.youtube.com/watch?v=_7KdeUIvlvw (2010)

事例5）製品開発の様子をオープンにしている

- 現在対応している不具合のリスト



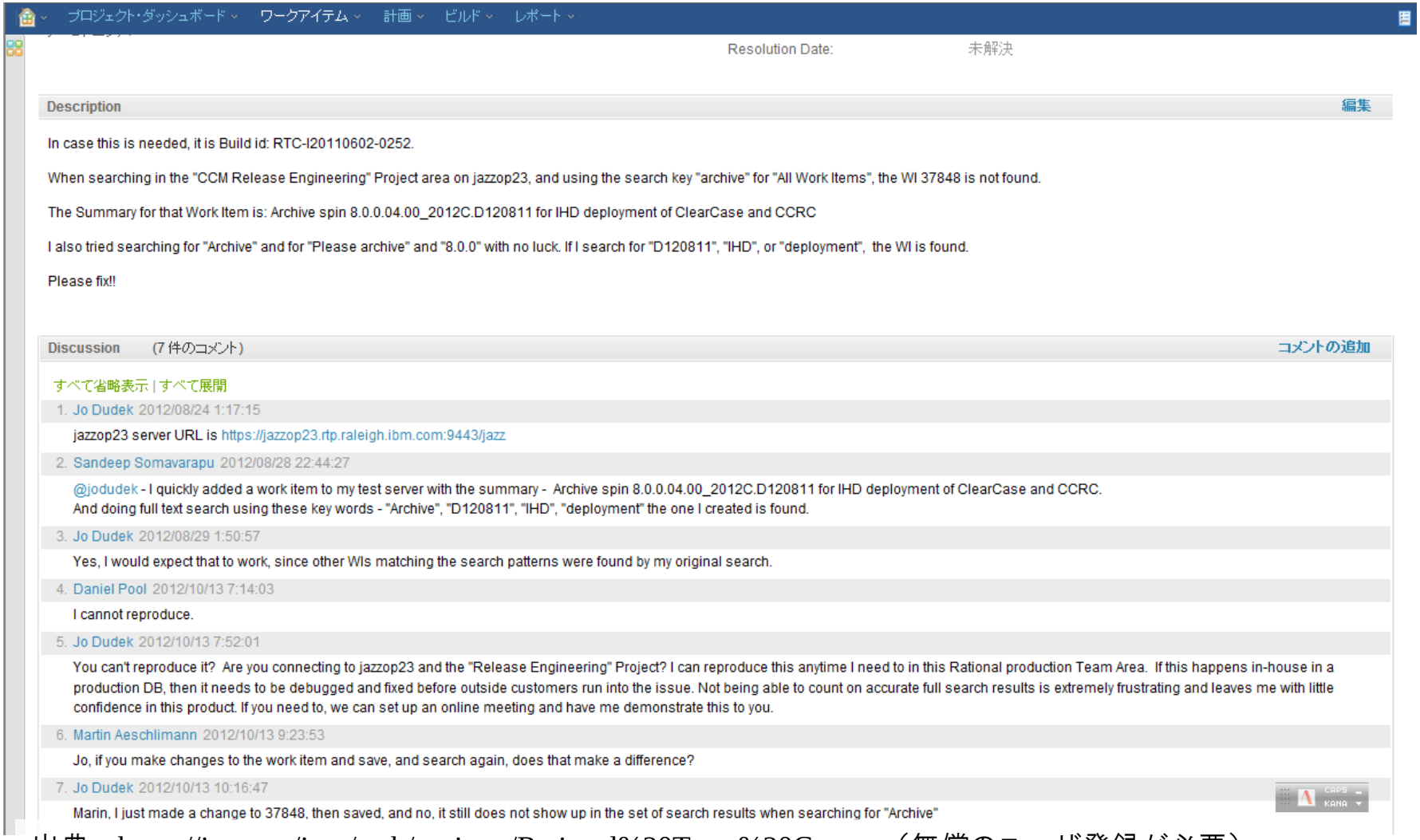
出典: https://jazz.net/jazz/web/projects/Rational%20Team%20Concert（無償のユーザ登録が必要）

事例5）製品開発の様子をオープンにしている

出典: https://jazz.net/jazz/web/projects/Rational%20Team%20Concert（無償のユーザ登録が必要）

事例5）製品開発の様子をオープンにしている

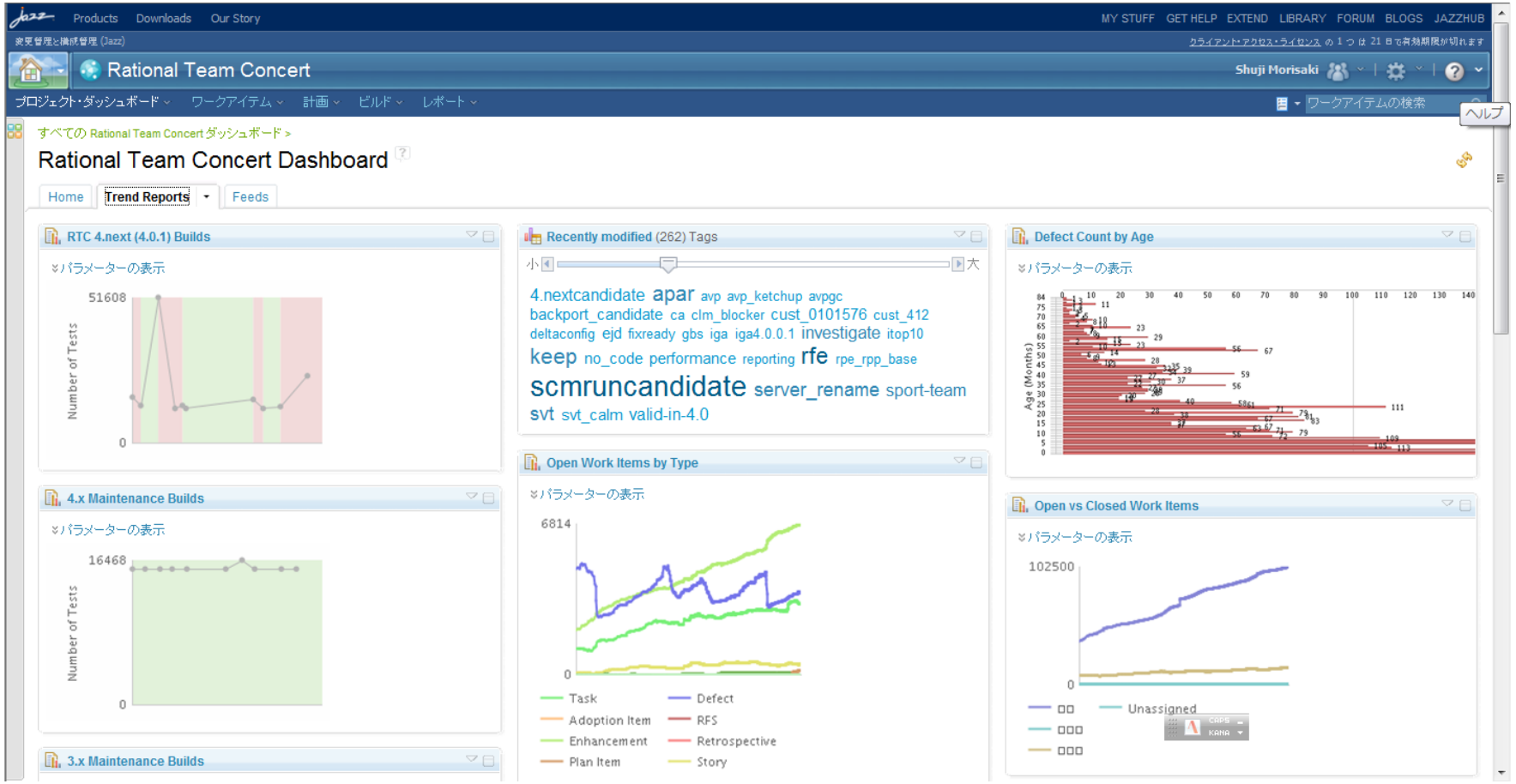
- 対応している不具合の詳細情報も閲覧できる。



出典: https://jazz.net/jazz/web/projects/Rational%20Team%20Concert（無償のユーザ登録が必要）

事例5）製品開発の様子をオープンにしている

- Rational Team Concert（開発支援環境）のダッシュボード（プロジェクト情報の集約）



出典: https://jazz.net/jazz/web/projects/Rational%20Team%20Concert（無償のユーザ登録が必要）

事例6）自動車のモデルベース開発

- 自動車の設計においてモデル化により短納期を実現する。
  - モデルでシミュレーションを十分に実施してから製作する。
  - 試作、実車での試行錯誤、手戻り、調整を省く。
  - 制御用ソフトウェアも同様にモデルベースで開発する。

出典: 原田 靖裕「ET2011基調講演よりSKYACTIVテクノロジーの誕生を支えたモデルベース開発～世界一のため、創造のためのモデルベース開発（MBD）へ～」, SEC Journal vol. 8, No.2, pp. 79-84(2012) http://sec.ipa.go.jp/sweipedia/catl033/catm047/cats047/56305/

事例6）自動車のモデルベース開発

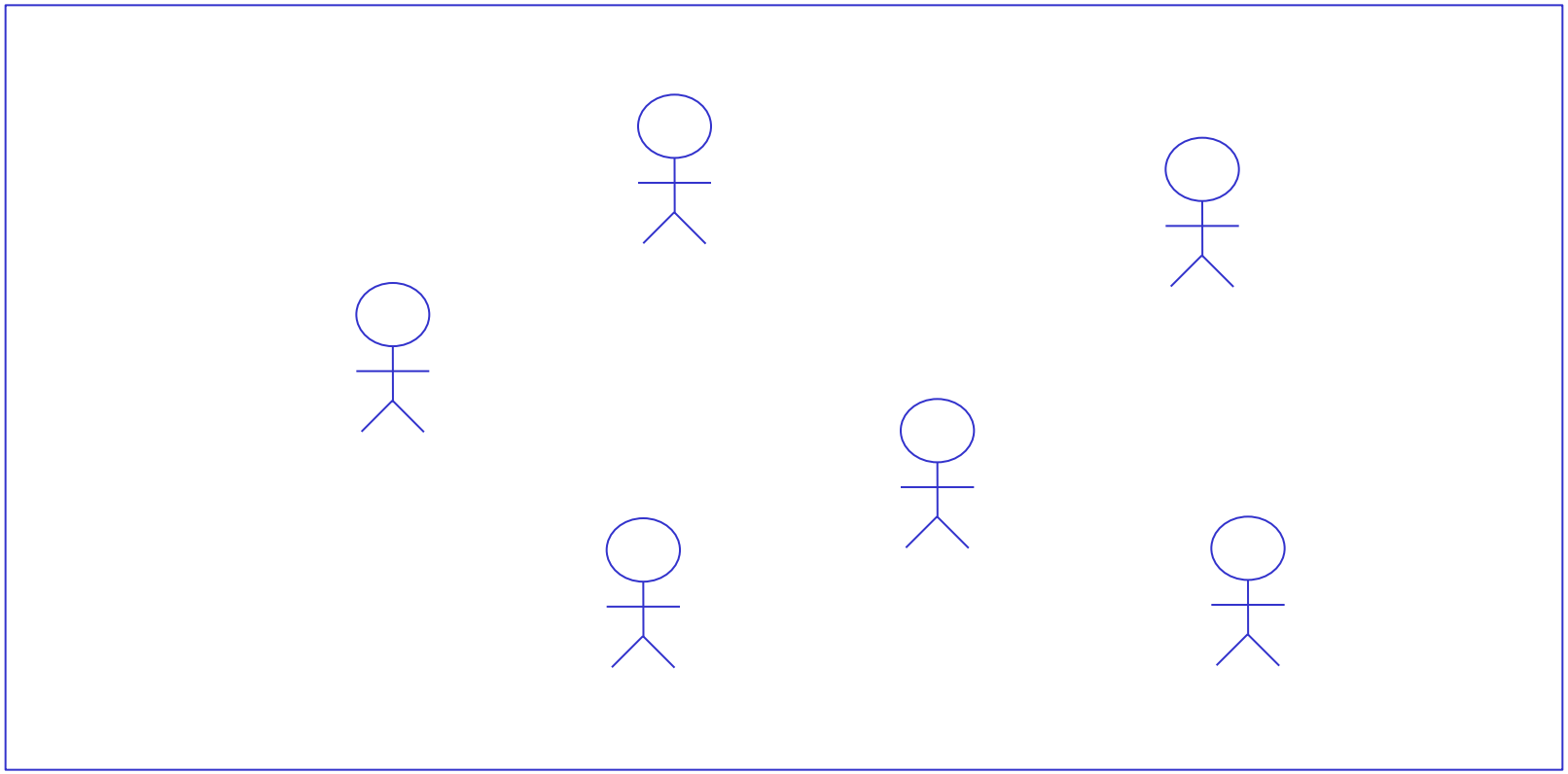
- レベル2に関連する記述
  - サーキットで最速のライン取りをドライバに模倣してもらい、他のドライバが最速と考えるラインよりも速く走れることを示した
  - 最初はモデルに懐疑的であった開発の最前線の専門家たちが「このモデルが無ければ、この燃焼は絶対に実現出来なかった」と考えを変える。
- レベル4に関連する記述
  - すぐに商品展開しない場合でも、技術準備はモデルを用いて着実に進めておき、いつでも展開出来る状態にしておくことが理想状態であると認識している。

出典: 原田 靖裕「ET2011基調講演よりSKYACTIVテクノロジーの誕生を支えたモデルベース開発～世界一のため、創造のためのモデルベース開発（MBD）へ～」, SEC Journal vol. 8, No.2, pp. 79-84(2012) http://sec.ipa.go.jp/sweipedia/catl033/catm047/cats047/56305/

アンチパターン（べからず集）

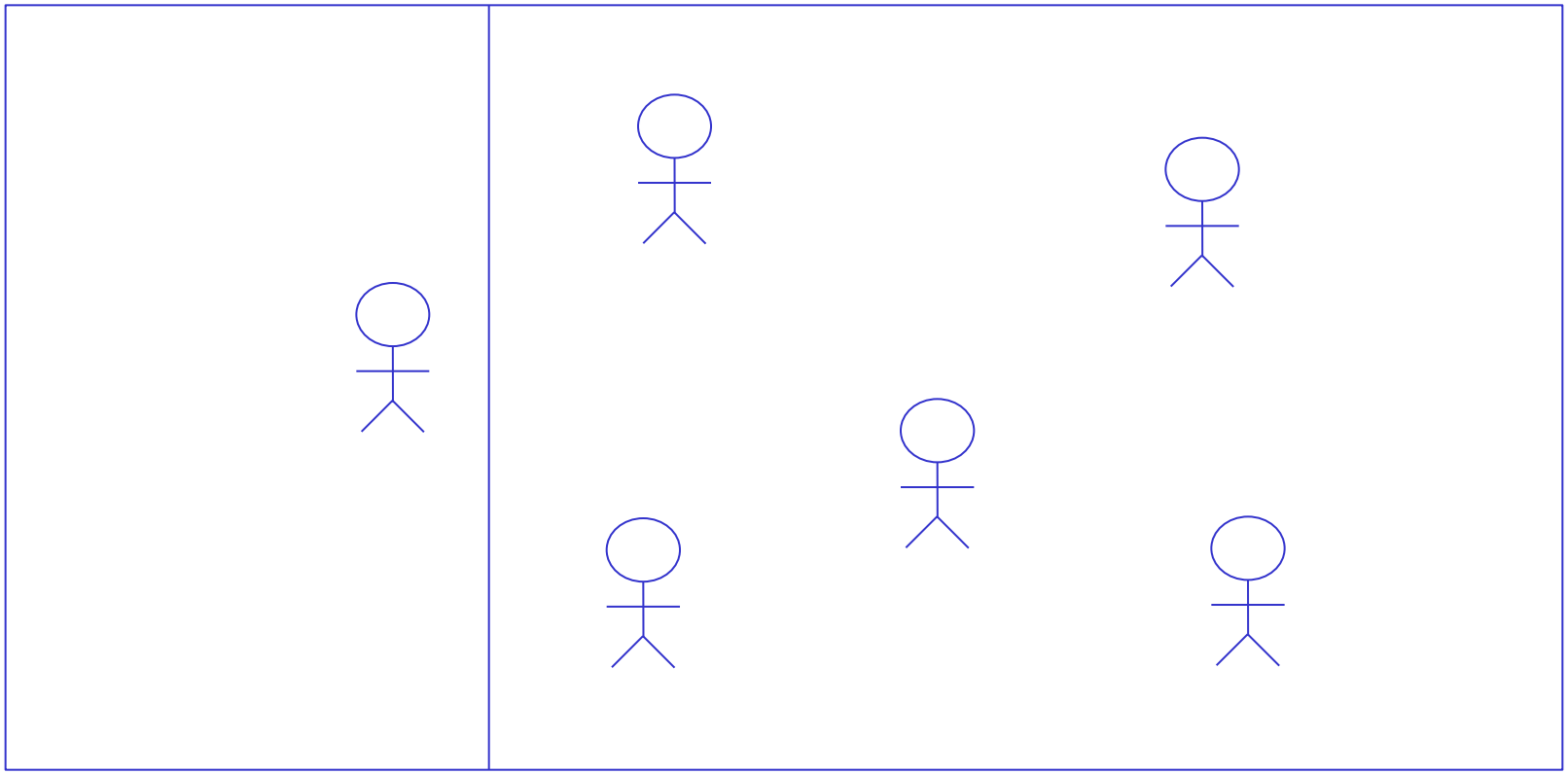
- 問題を見つけて終わりにする。
  - 「～だからダメなんだ。ウチは」と根拠をみつける。
  - 「～ヤツがいるからダメなんだ」と線引きする。
- ルール変更の余地を考えず「ルールだから」で終わり。
- 事例、知見、技術を何らかの形で活かさず終わり。

アンチパターン：問題をみつけて終わりにする



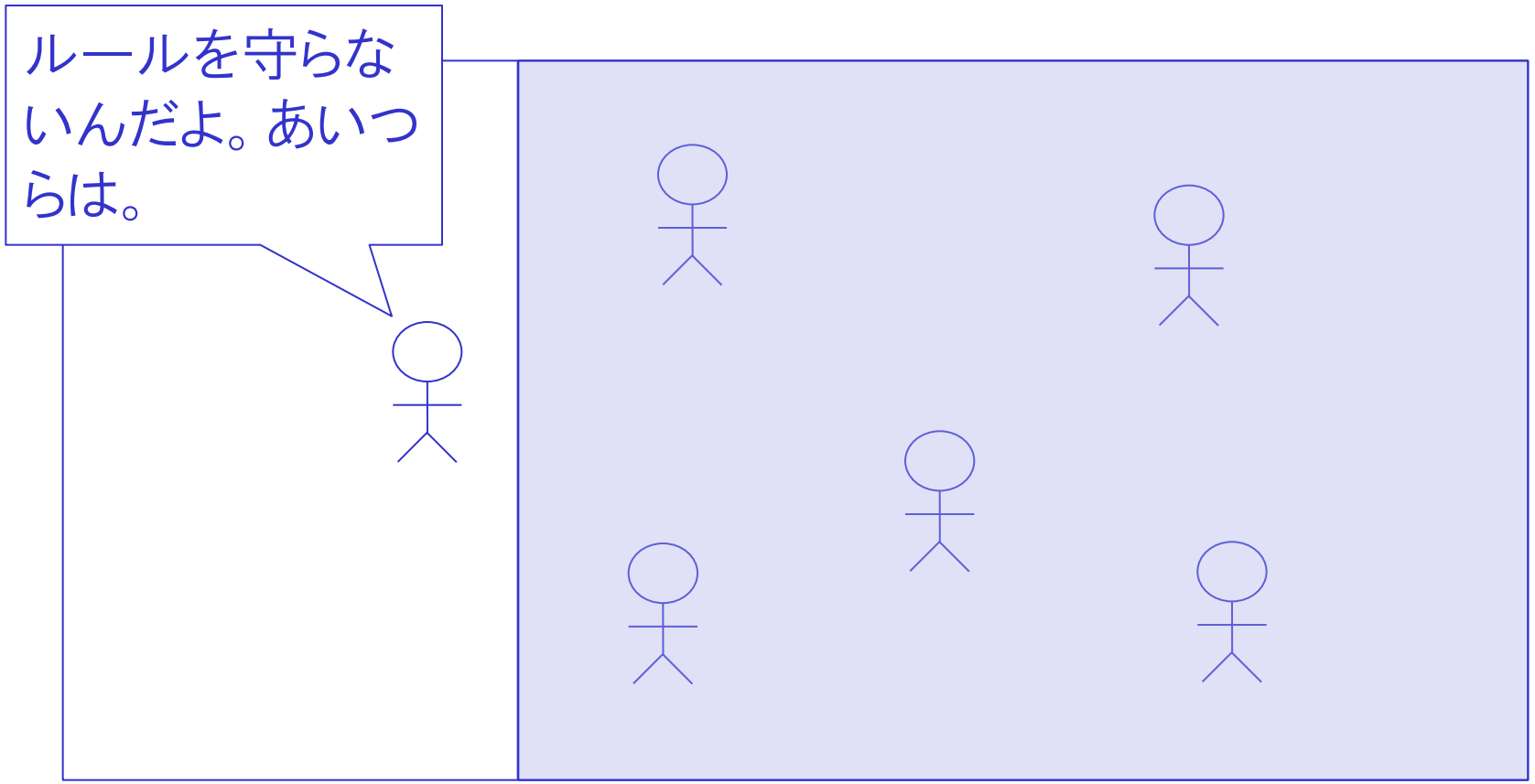
アンチパターン：問題をみつけて終わりにする

- ① 線を引く。



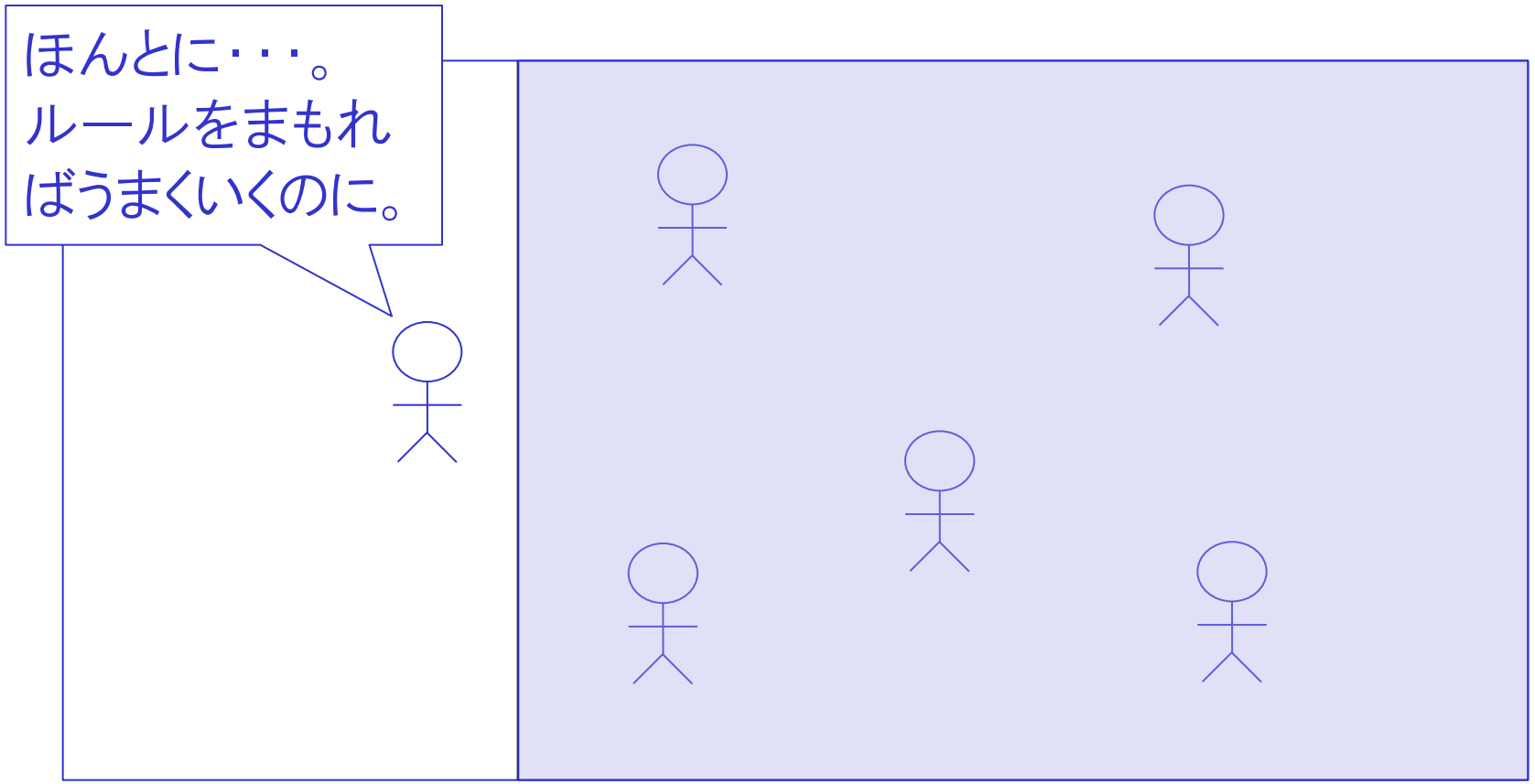
アンチパターン：問題をみつけて終わりにする

- ② 自分がいないほうの欠点を指摘し、批判をする。



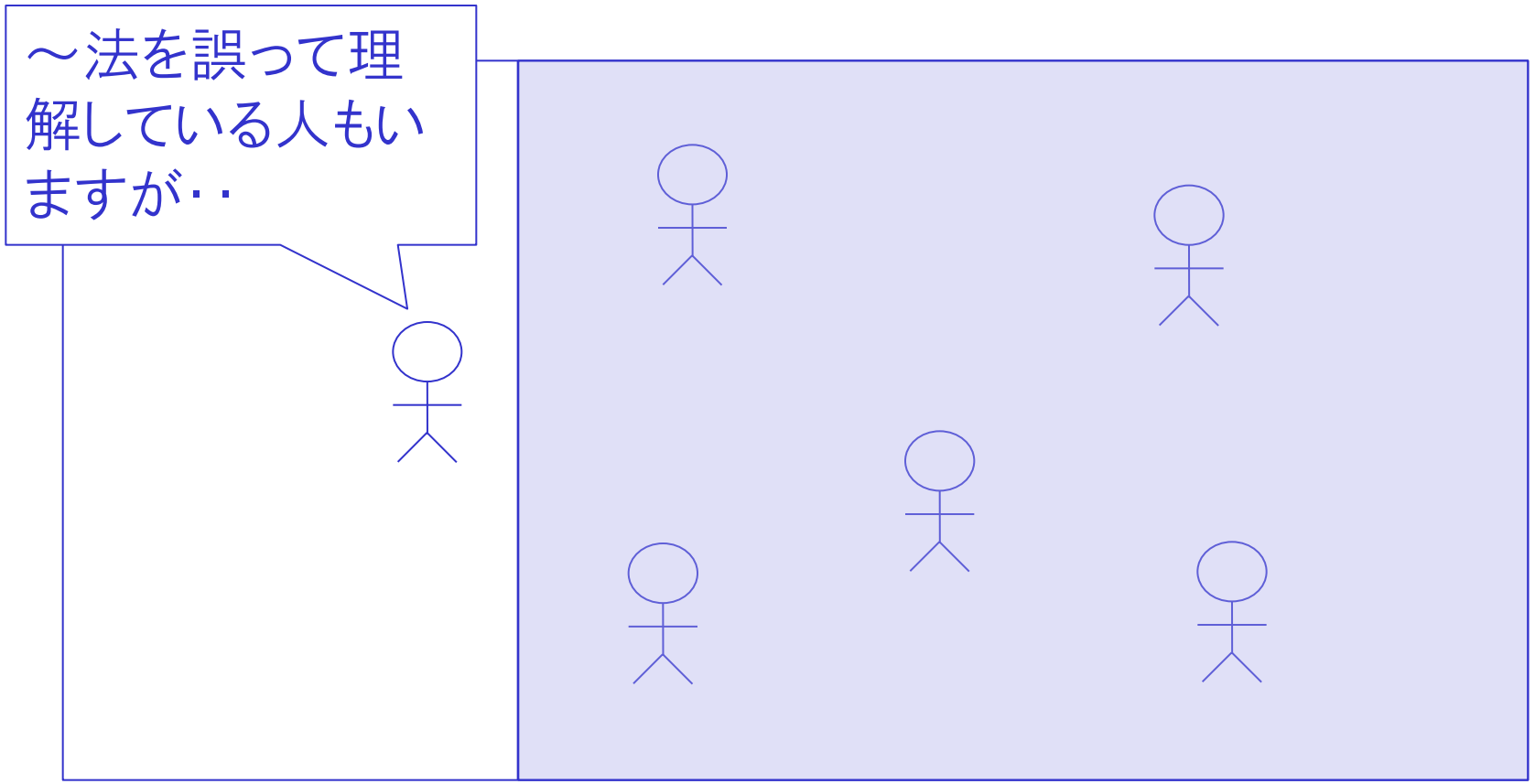
アンチパターン：問題をみつけて終わりにする

- ③ 安心してすっきりして、終わり。



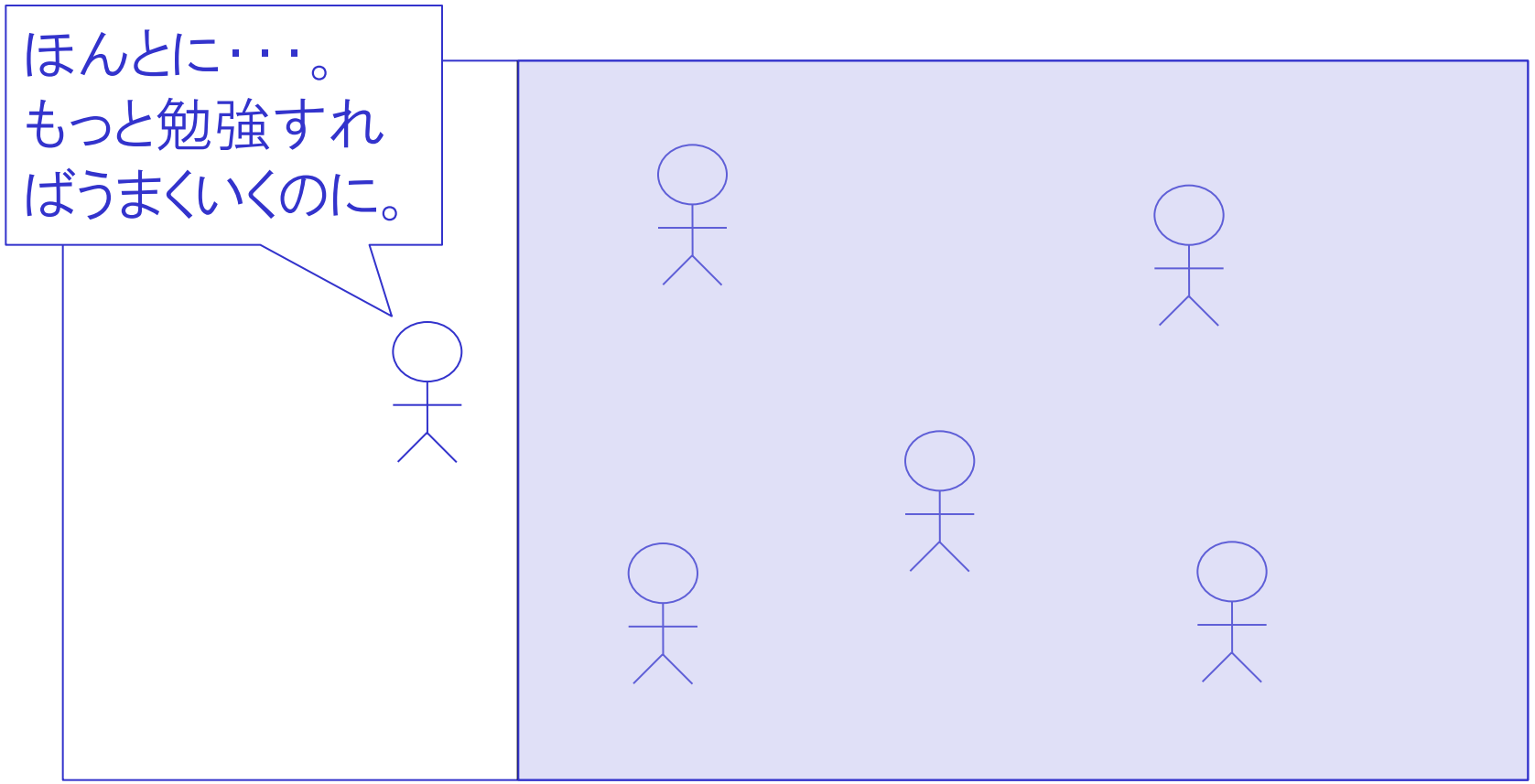
アンチパターン：問題をみつけて終わりにする

- ② 自分がいないほうの欠点を指摘し、批判をする。



アンチパターン：問題をみつけて終わりにする

- ③ 安心してすっきりして、終わり。



まとめ

- 私の経験にもとづき4つのレベルに分解した。
- 品質向上活動には複数のレベルがあり、単純に技術・技法の延長線だけではうまくいかないことがある。
  - レベル1 → 2: 合意
  - レベル2 → 3: 他との比較
  - レベル3 → 4: 競争力の分析
- 他の事例を聞いてマネするだけではうまくいかない。
- 普段の活動を定期的に見直し、本当に品質向上につながっているか振返る。
- 参考情報
  - twitter: @smorisaki
  - ブログ: http://blogs.itmedia.co.jp/morisaki/

お知らせ

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。

2. 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。