Università degli Studi dell'Insubria

Lessons learned on software maintenance: any relief at horizon?

Luigi Lavazza

Dipartimento di Scienze Teoriche e Applicate

Università degli Studi dell'Insubria

Varese, Italy

Maintenance: what to measure?

- What am I paying for?
- Measures are needed to relate maintenance costs to maintenance activities.
- Maintenance
  - An activity
    - The trousers analogy
  - Maintenance vs. reuse
    - The analogy does not hold any longer

What is currently measured (Functional size measurement methods)

Functions made available via the GUI:

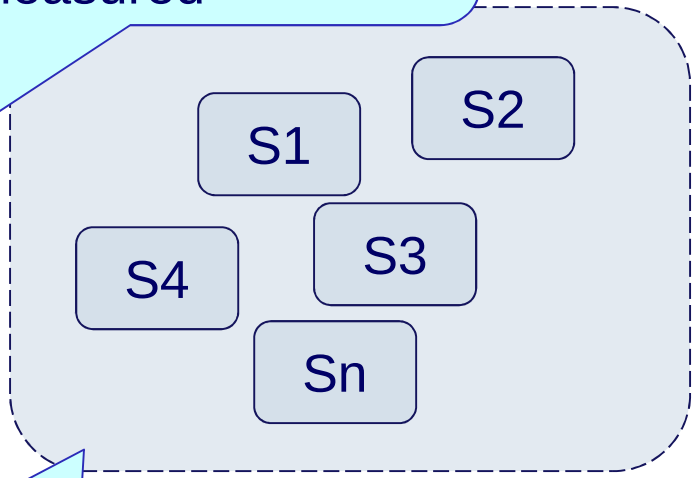
✓ measured

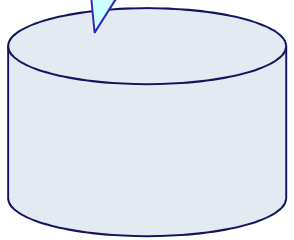
Data entering/exiting the application:

✓ measured

Data managed by the application:

✓ measured



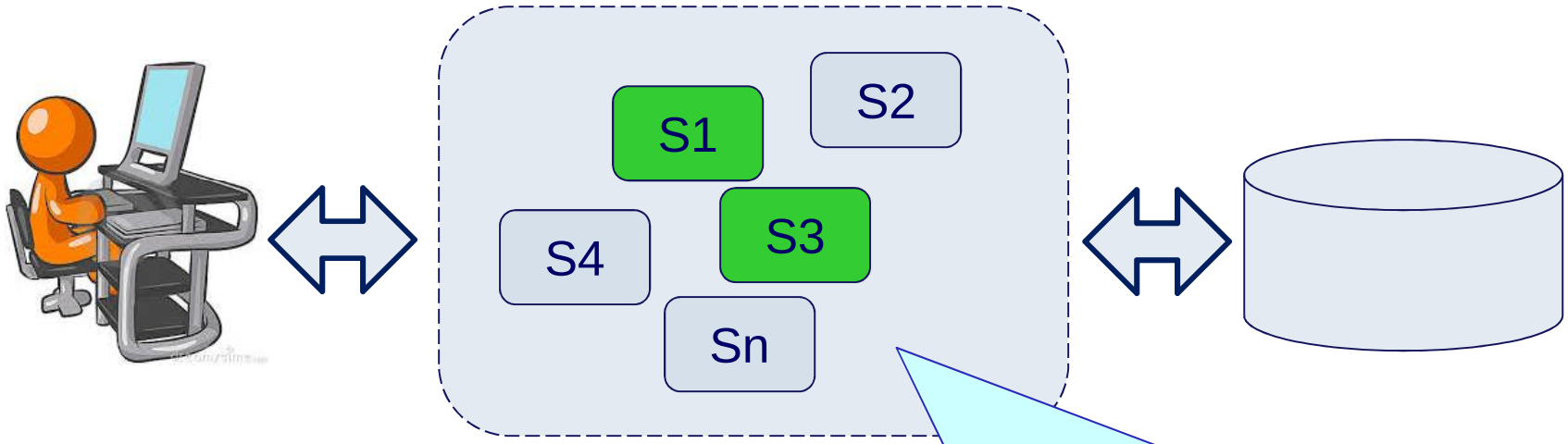


Services and components used to implement the application:

✗ not measured

Problems

- Reusability is always (to some extent) there, even when not strictly required.

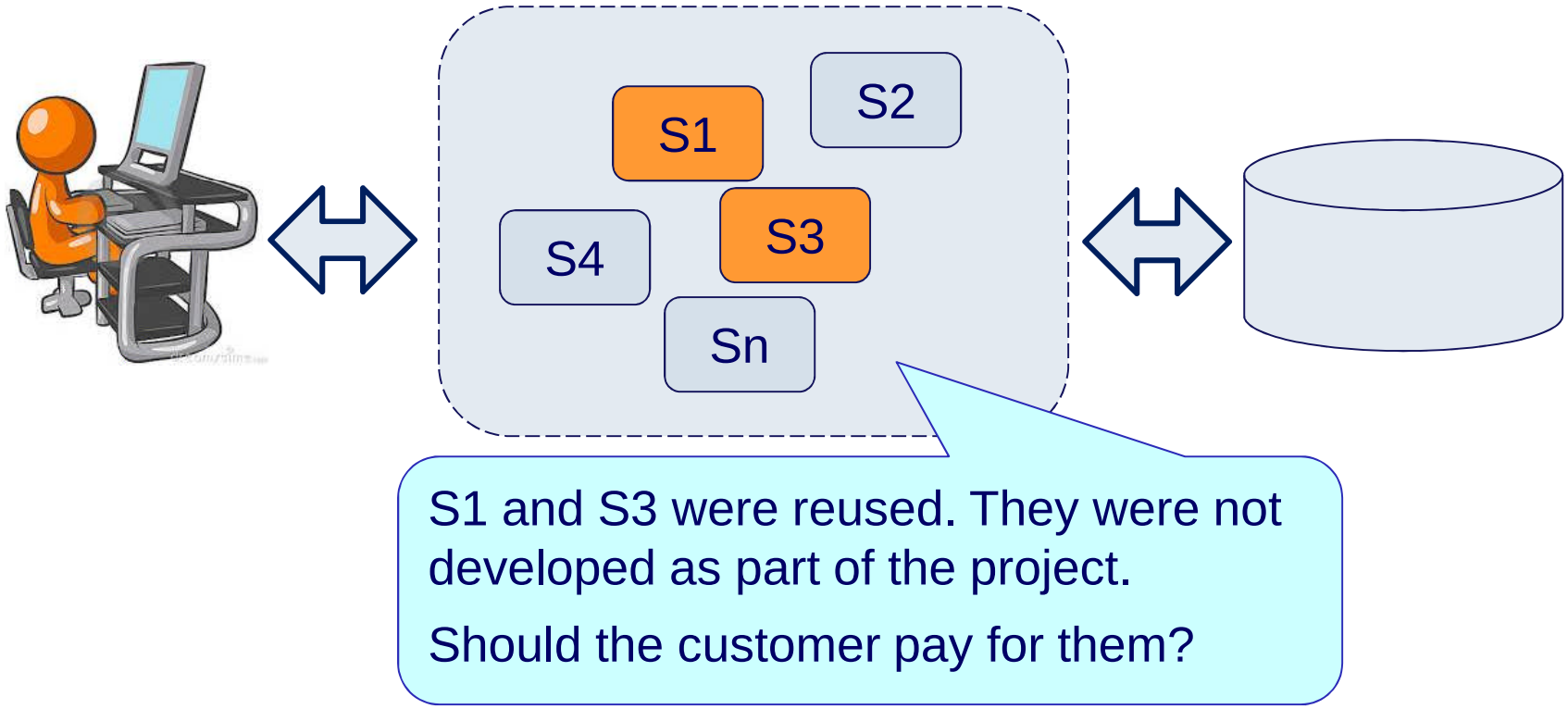


S1 and S3 were developed as part of the project and are reusable.

Are they an additional asset for which the developer should be paid?

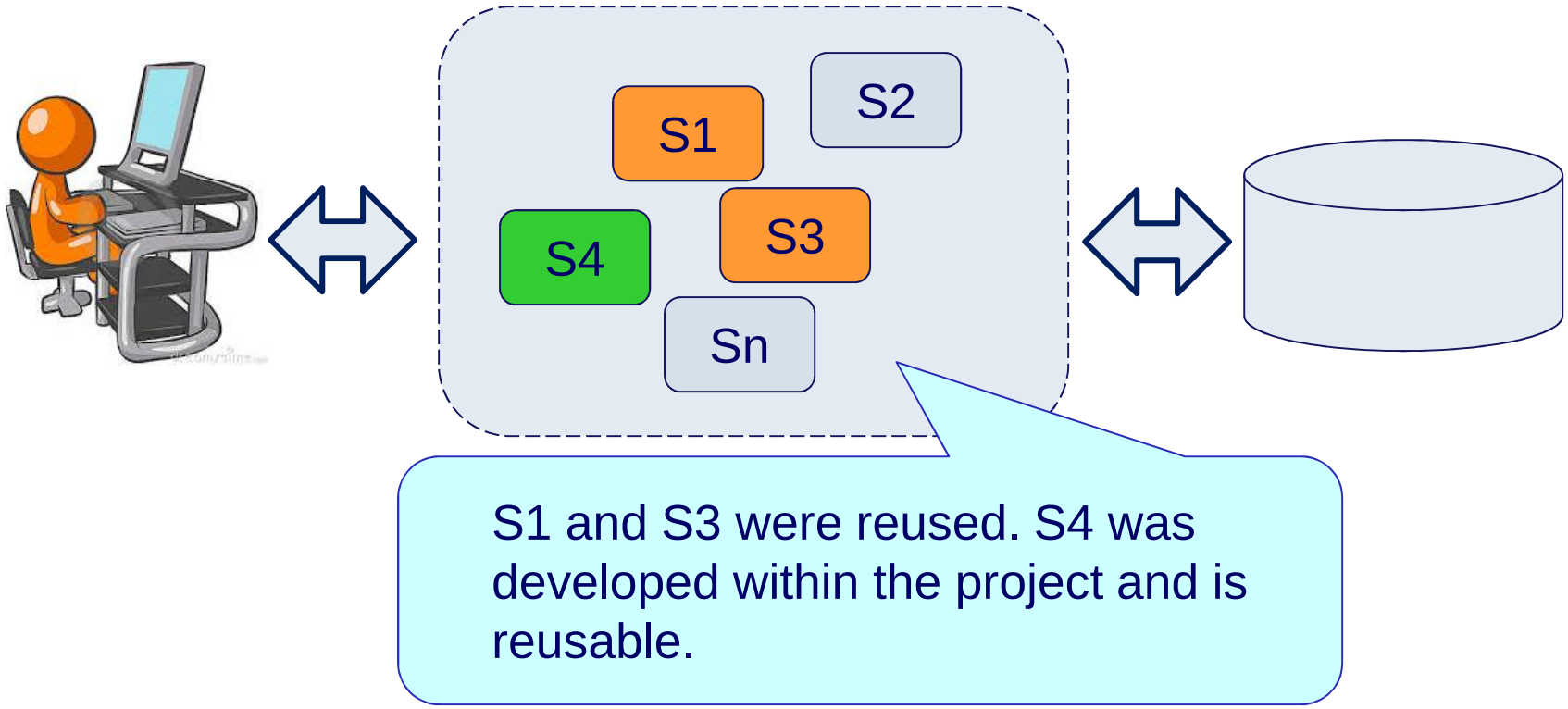
Problems

- Reuse is not measured by current FSM methods.



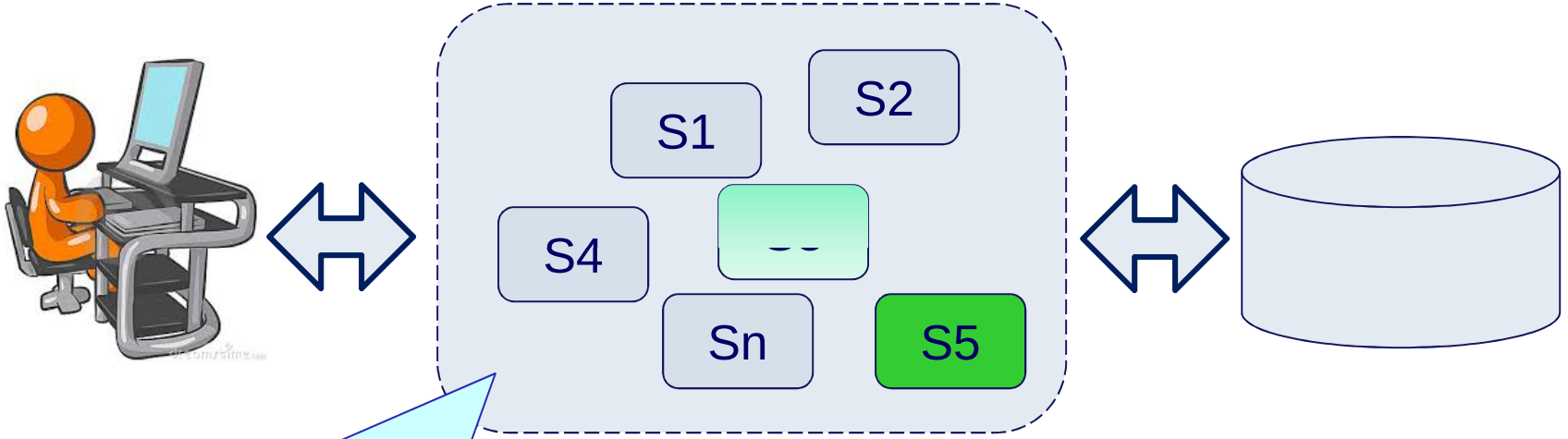
Problems

- The mixed case



Problems

- Maintenance
  - The project is conceived as a maintenance project
  - The size is measured at the interface/logical data level



S3 is modified. S5 is newly developed as part of the project.

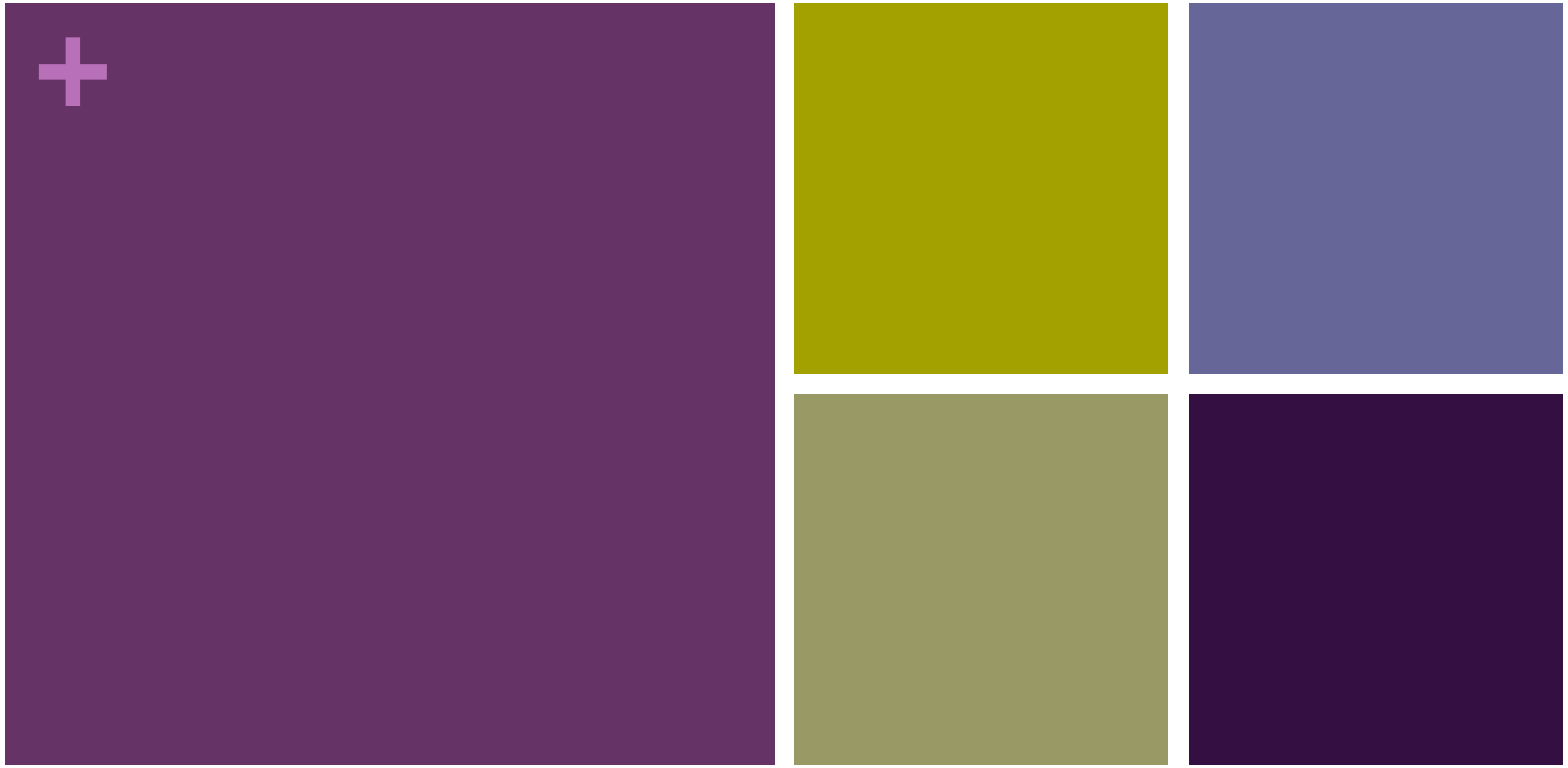
With current FSM methods, the size of the maintenance depends on how many user-visible functions depend on S3 and S5.

A possible solution

- Separate what is achieved from what is done.
- What is achieved:
  - New functionality
  - New reusable assets

Current FSM consider only this aspect.

- What is done:
  - Components/services modified
  - Components/services added
    - Size and *complexity* of the modifications/additions could be measured
- The result of the measurement should be a *vector* of measures.

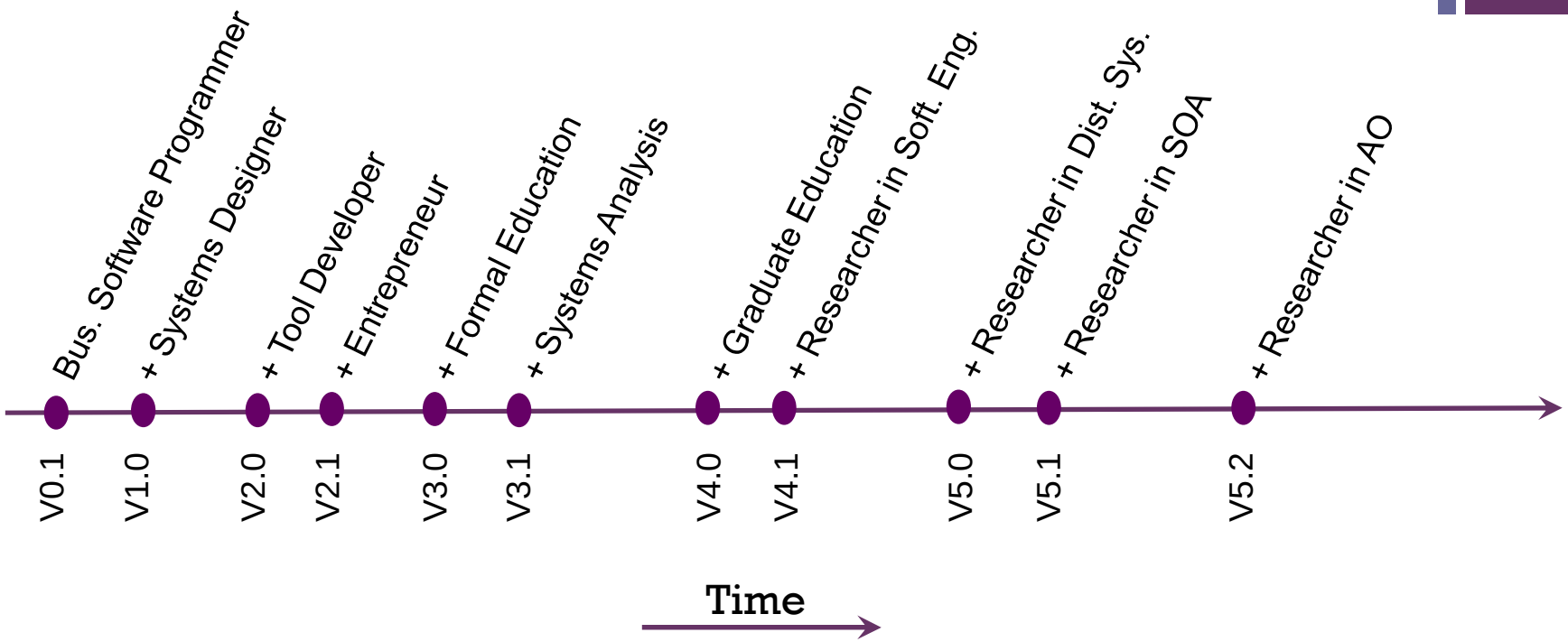


LESSONS LEARNED IN SOFTWARE MAINTENANCE : Any Relief on the Horizon?

Dr. Stephen W. Clyde (swc)
Utah State University

ICSEA 2014
OCTOBER 14, 2014

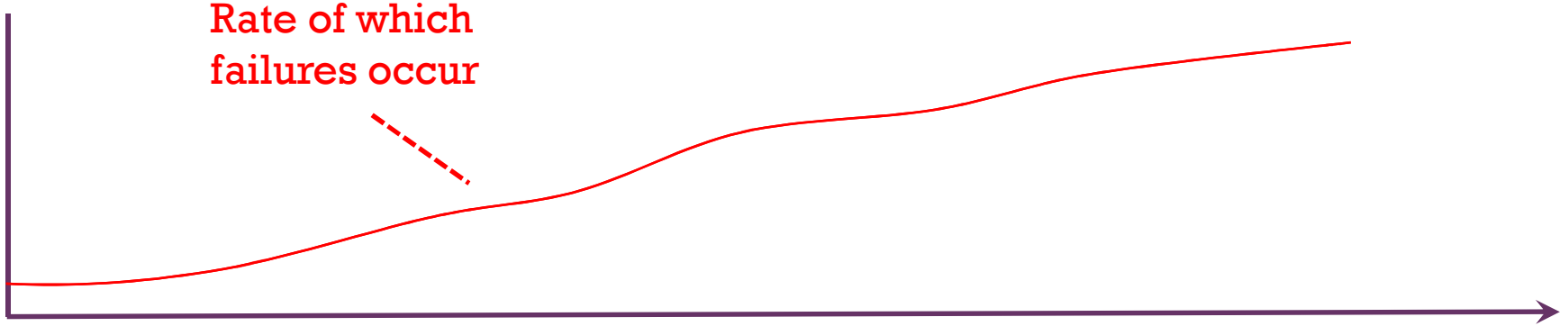
swc's Maintenance & Enhancement Life Cycle



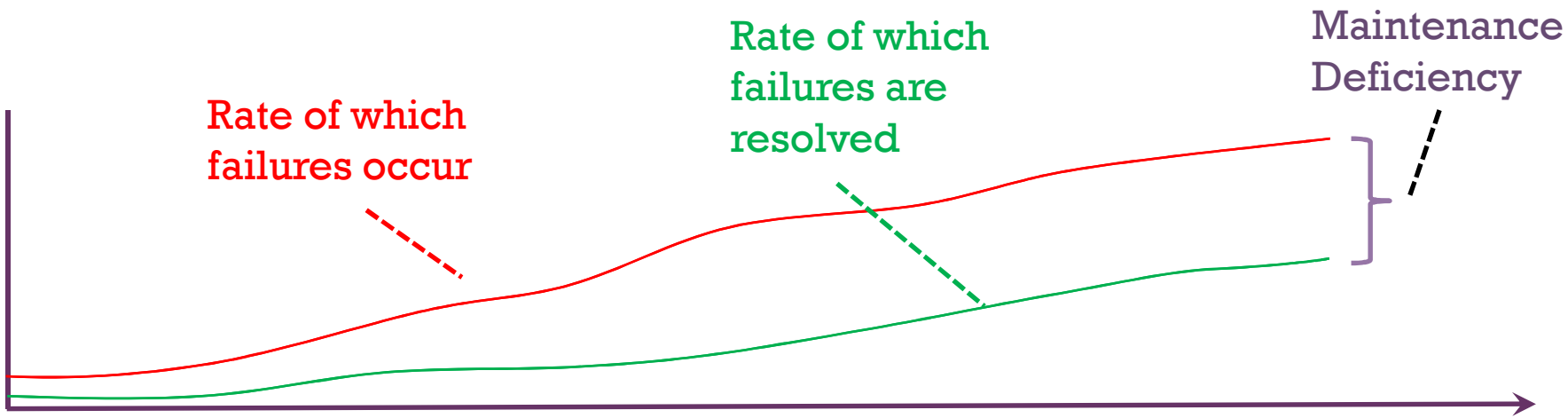
(sorry, no actual dates; they are too scary)

Continuous Integration

swc's Maintenance & Enhancement Life Cycle



swc's Maintenance & Enhancement Life Cycle

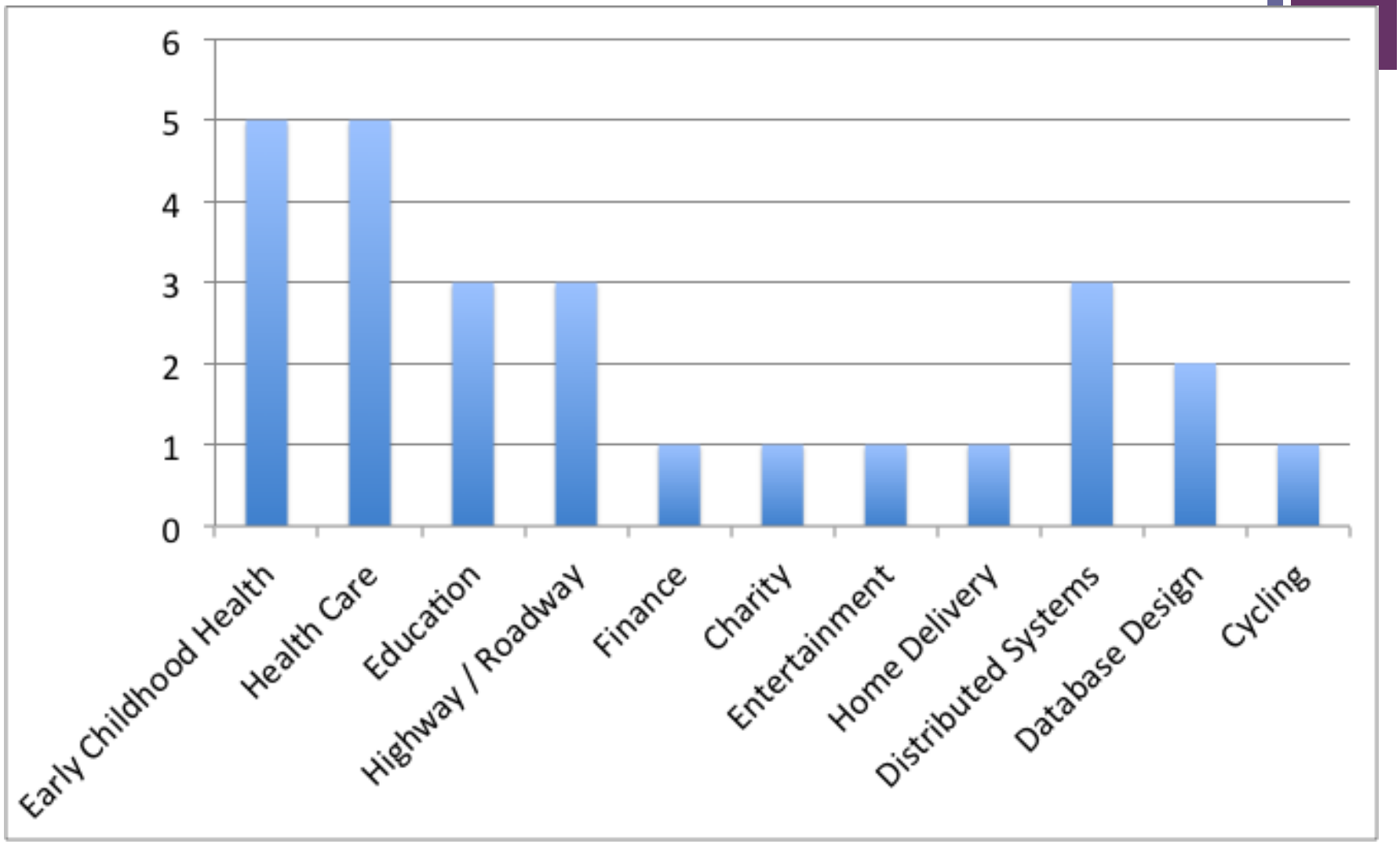


Antidotal Evidence on Software Maintenance

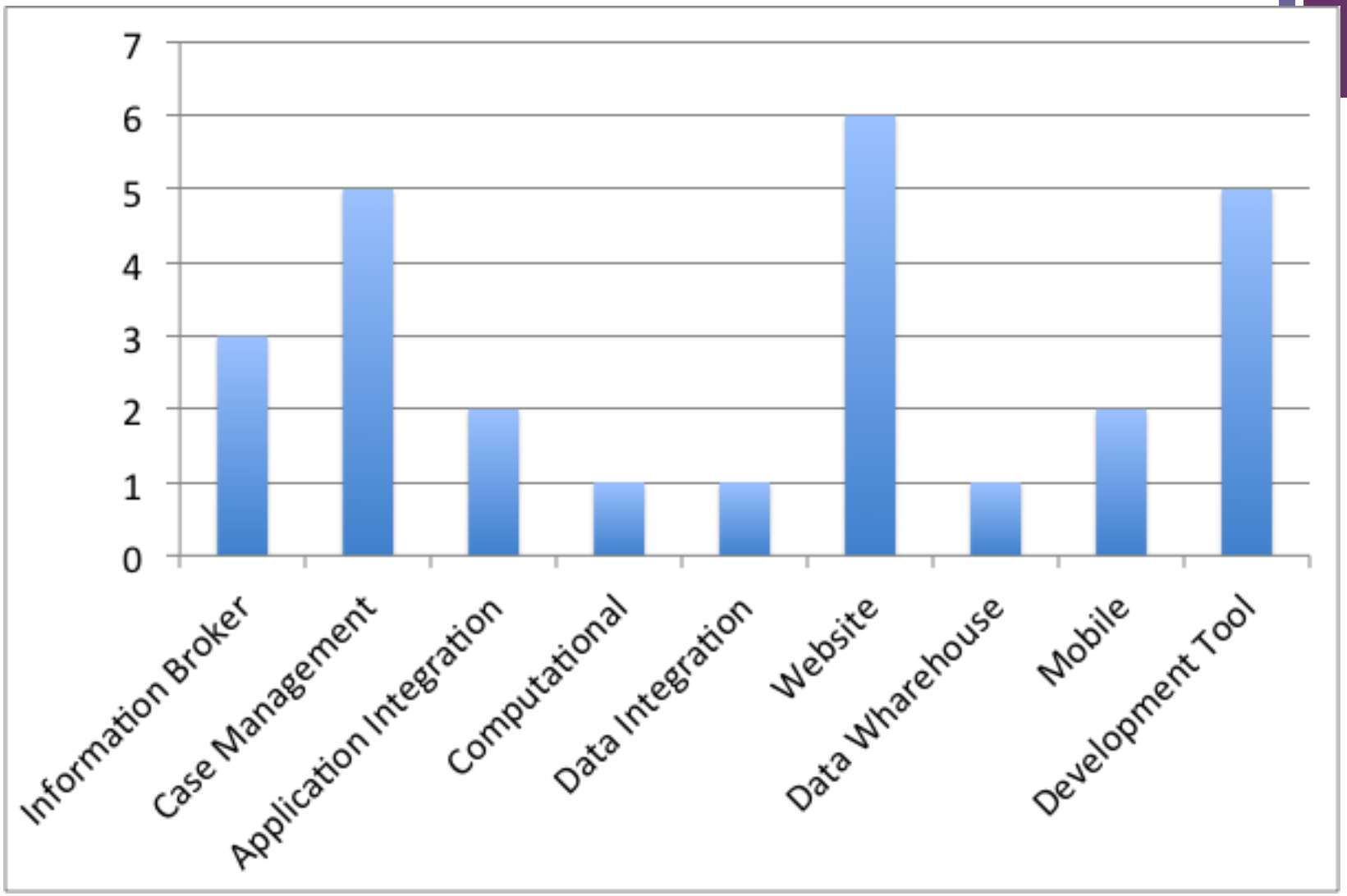
(from a very informal, non-scientific study)

- Examined a Pool of 26 projects
  - Commercial-grade, built-to-suite projects (real customers, real needs, low tolerance for bad software)
  - In development and/or maintenance 2000-2014
  - Significant personal involvement as project lead, technical lead, consultant, or developer.
  - Significant software developer hours
- Maintenance = Bug Fixes, Upgrades, and Enhancements

Application Domains



Software System Types



+ Development / Maintenance Years

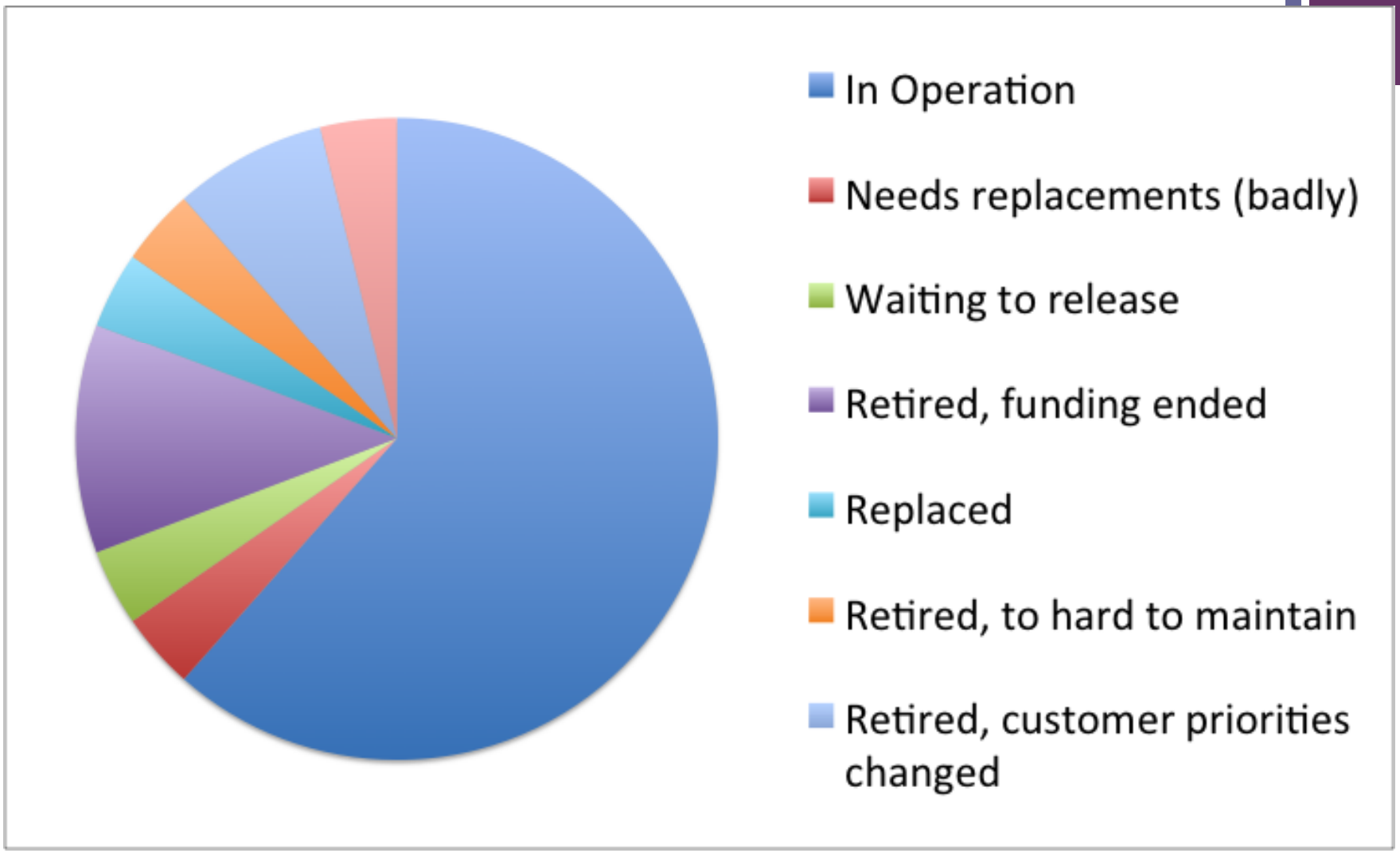
Years to 1st Release

Statistic	Years
Minimum	0.5
Median	0.5
Average	0.96
Maximum	3

Years of Maintenance

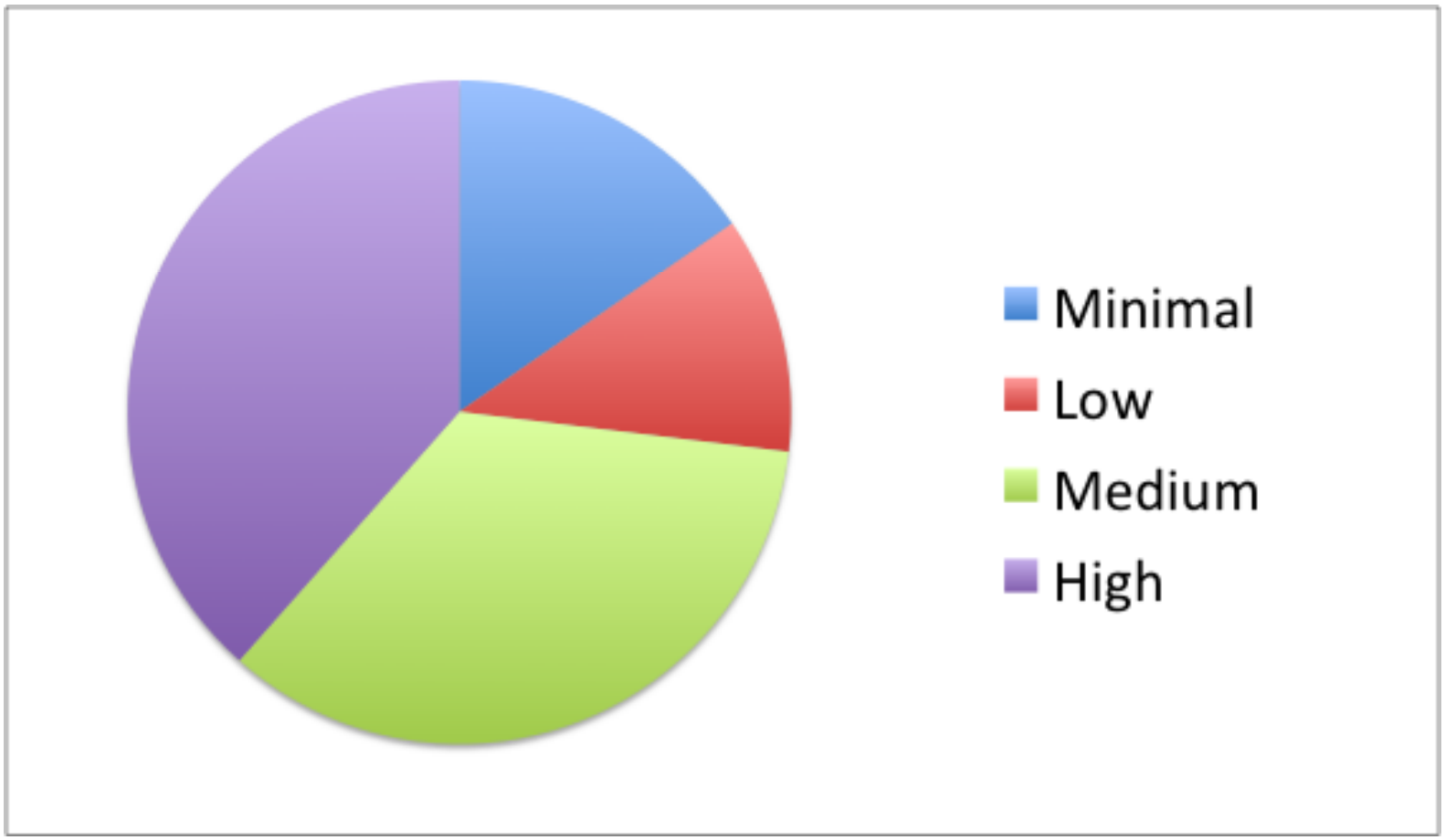
Statistic	Years
Minimum	0.1
Median	4
Average	5.4
Maximum	30

Current Status



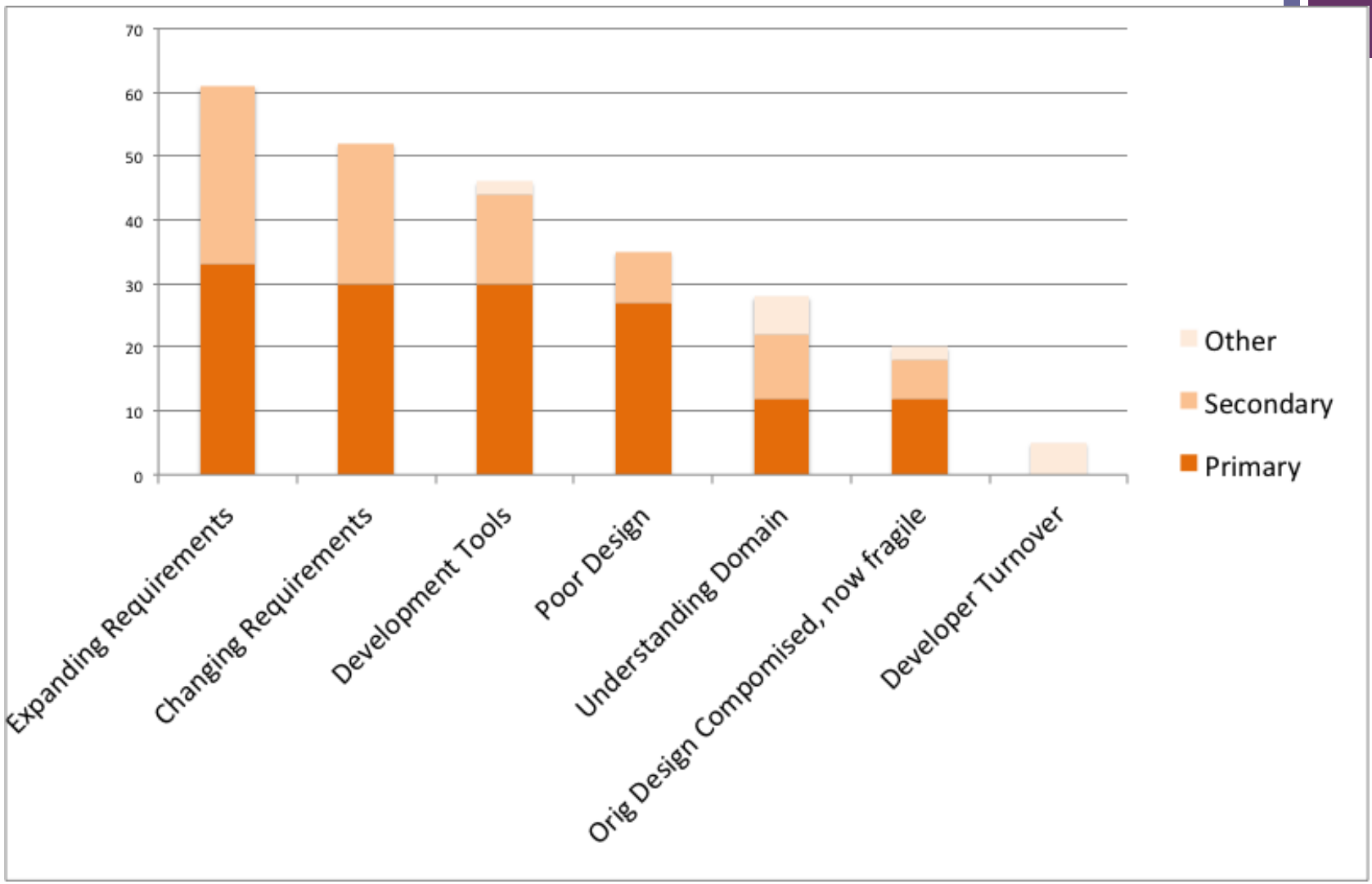
Maintenance Severity – Pain

(subjective measurement)

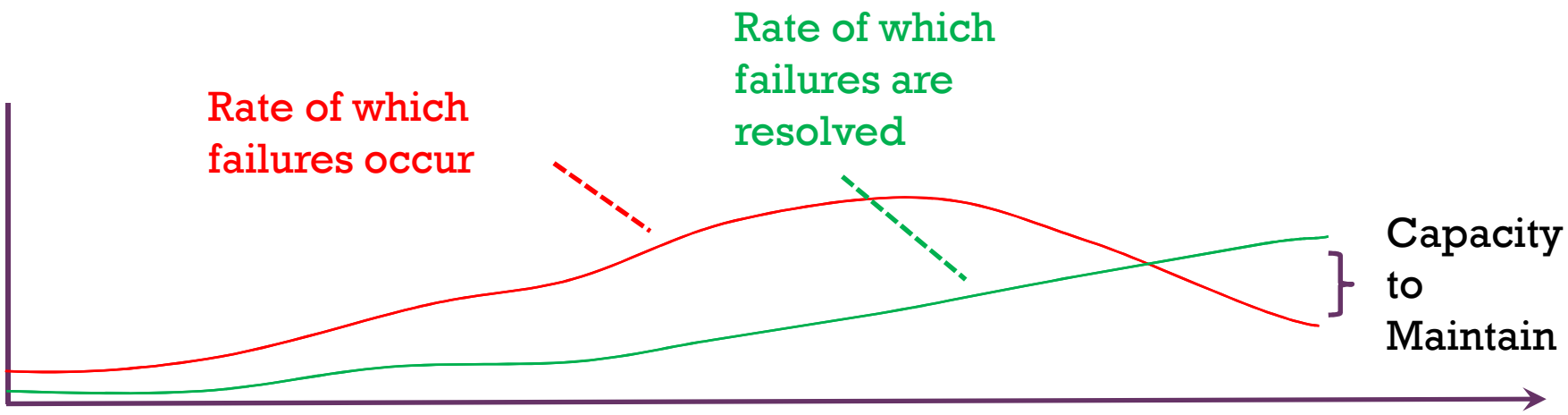


Maintenance Issues

(weighted from top three and by severity)



Creating a Capacity to Maintain (or evolve) Software Systems



- We have to both:
  - Reduce the rate at which failure (or requests for new/change features) occur
  - Increase our ability to resolve such issues quickly

How Do we Improve Our Maintenance Capacity

- Anticipate or accommodate new or changing requirements
  - Better designs, with better separation of concerns
    - Aspect Orientation can help, particular when using high-level aspects
  - Better anticipation on potential "bend" points in the software
- Choose development tools carefully; change only if truly justified
- Better Designs
  - Flexible architectures, like service-oriented architectures
  - Adoption/Adaptation of appropriate design patterns

Is Any Relief on the Horizon?

- Yes, but it dependents on us
  - Individually, and
  - Collectively
- Don't expect relief to come from new tools only
- Relief will come from disciplined application of what we know at the time

FUJITSU

shaping tomorrow with you

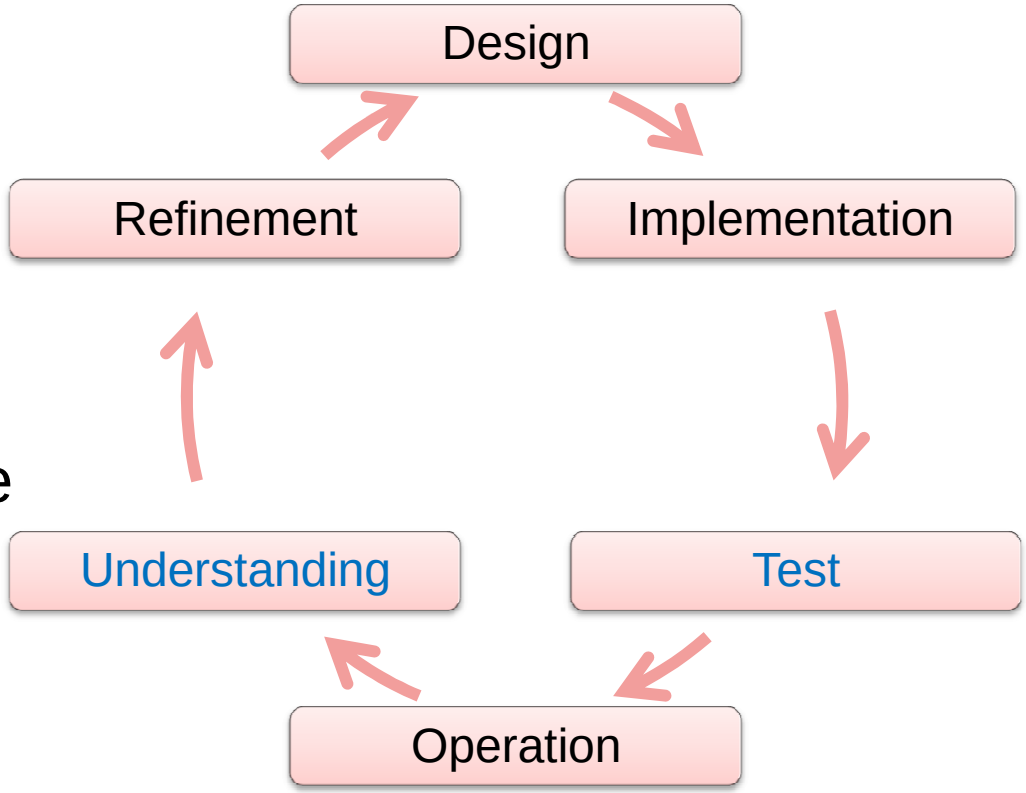
Panel Discussion "Lessons Learned on Software Maintenance: Any Relief at Horizon?

Hideo Tanida
Software Engineering Laboratory
Fujitsu Laboratories Ltd., Japan

Software Development Cycles

FUJITSU

- Software development has CYCLES
  (≠ FLOW in waterfall model)
- "Software maintenance" can be considered a term referring to the whole cycle
  - Esp. in iterative development styles such as Agile development

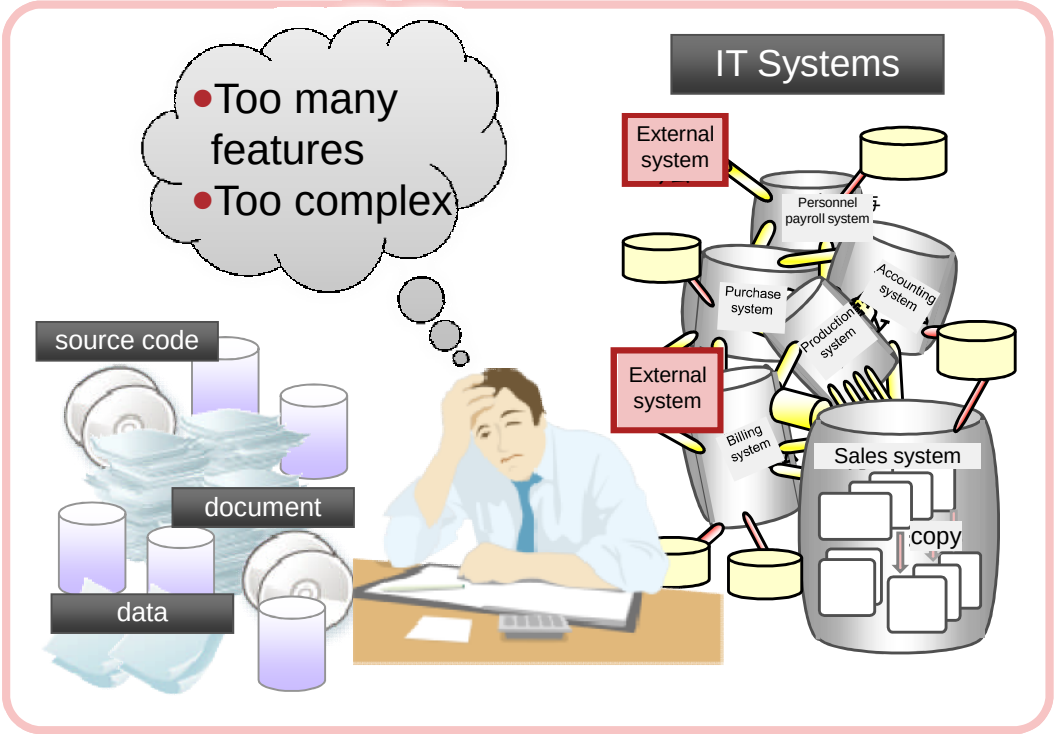


- We introduce two technologies for "Understanding" and "Test"
  - Are the technologies of *any relief at horizon*?

Need for Support in Understanding Code

FUJITSU

- Maintaining and enhancing **large** and **long-lived (10+ years)** IT systems are very difficult challenges.
  - Increasing **features**, specifications, functionalities, and requirements
  - Increasing complexity
  - Knowledge loss
- Rapid Understanding of IT systems is required.
  - Overall structure
  - What features exist



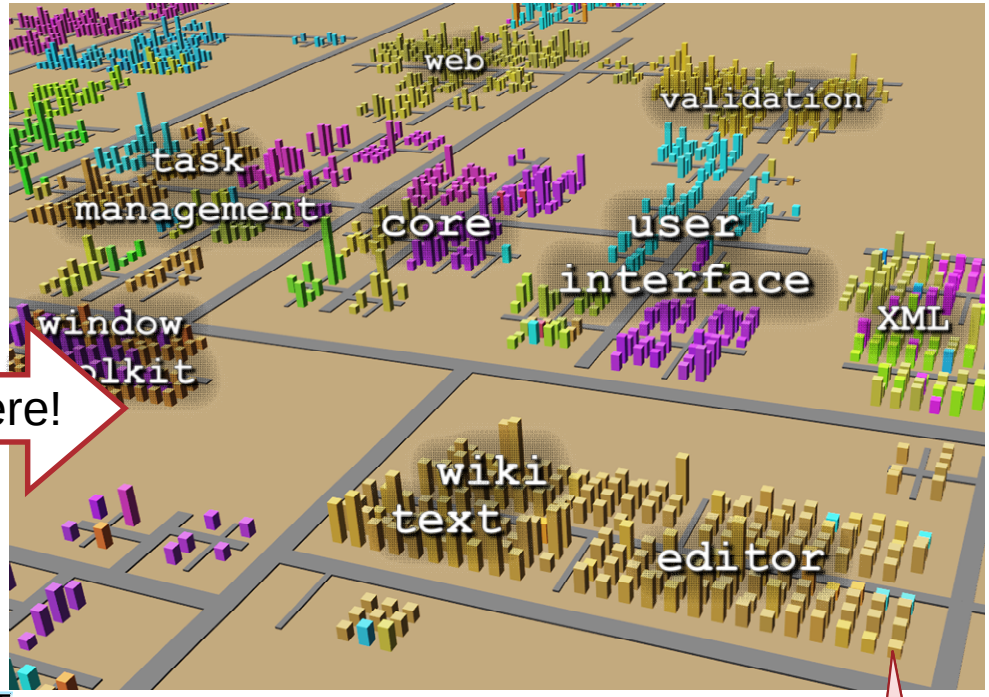
Software Map Technology

FUJITSU

enables rapid understanding of IT systems.

- Overall structure of the system
- What features exist in the system?
- What source files are involved in each feature?
- Current status of the features

Software Map



are here!

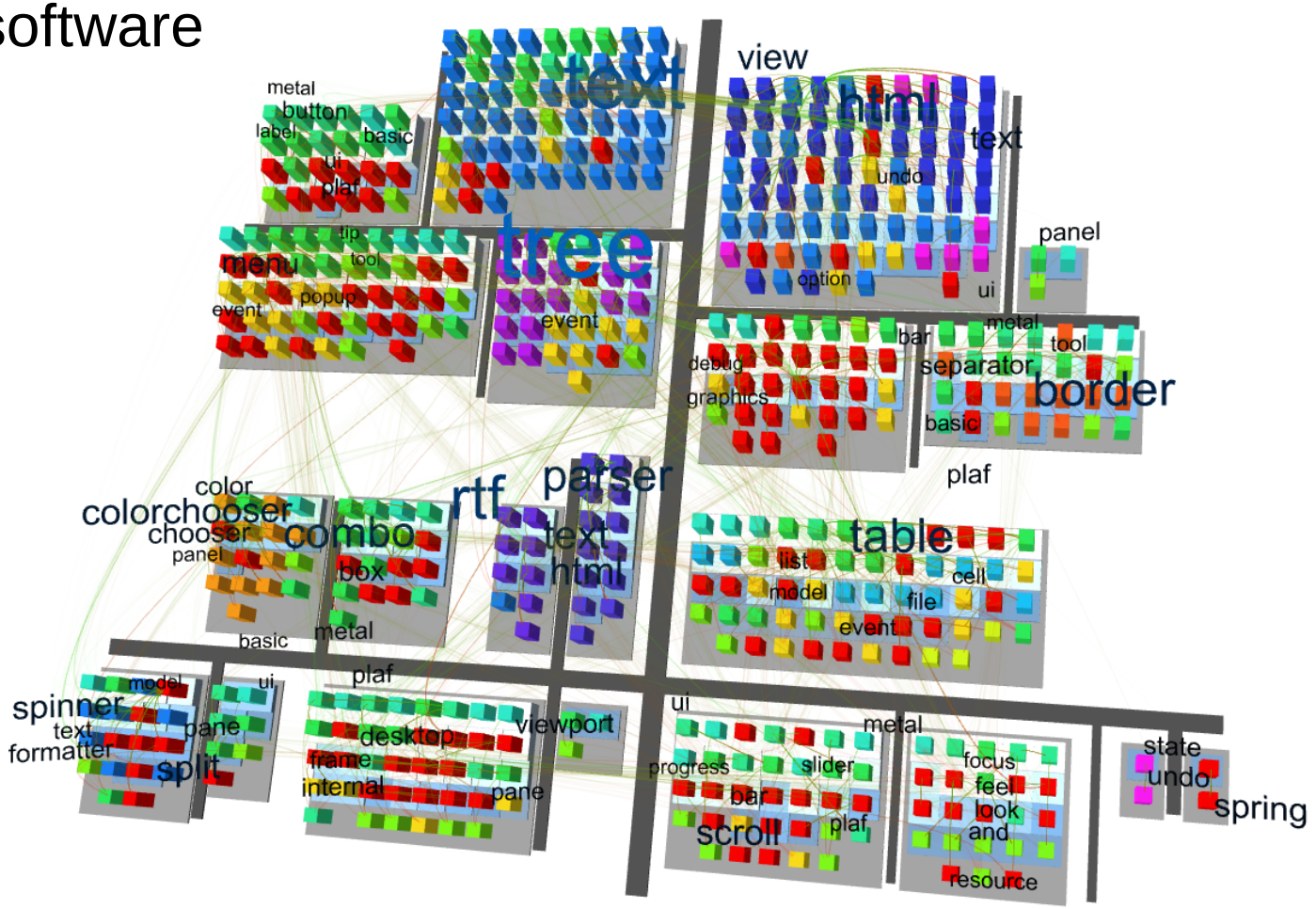
Building = source file (class)

Software Map also enables important analyses:

Analysis on JDK Swing 1.4.0 (536 classes)

FUJITSU

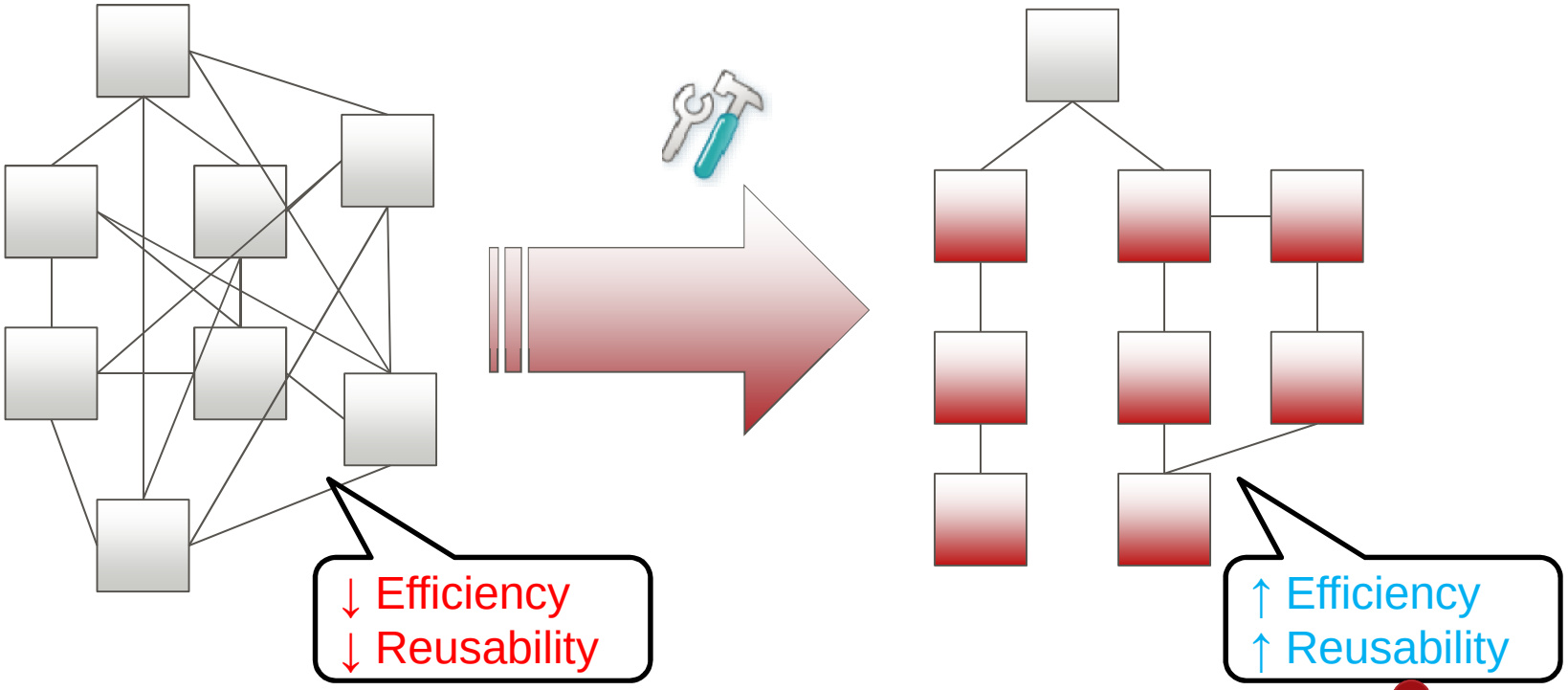
- We are successfully extracting features, layers, and architectural knowledge of target software



Need for Support in Compatibility Testing

FUJITSU

- Software evolves continuously with fixing and adding new features

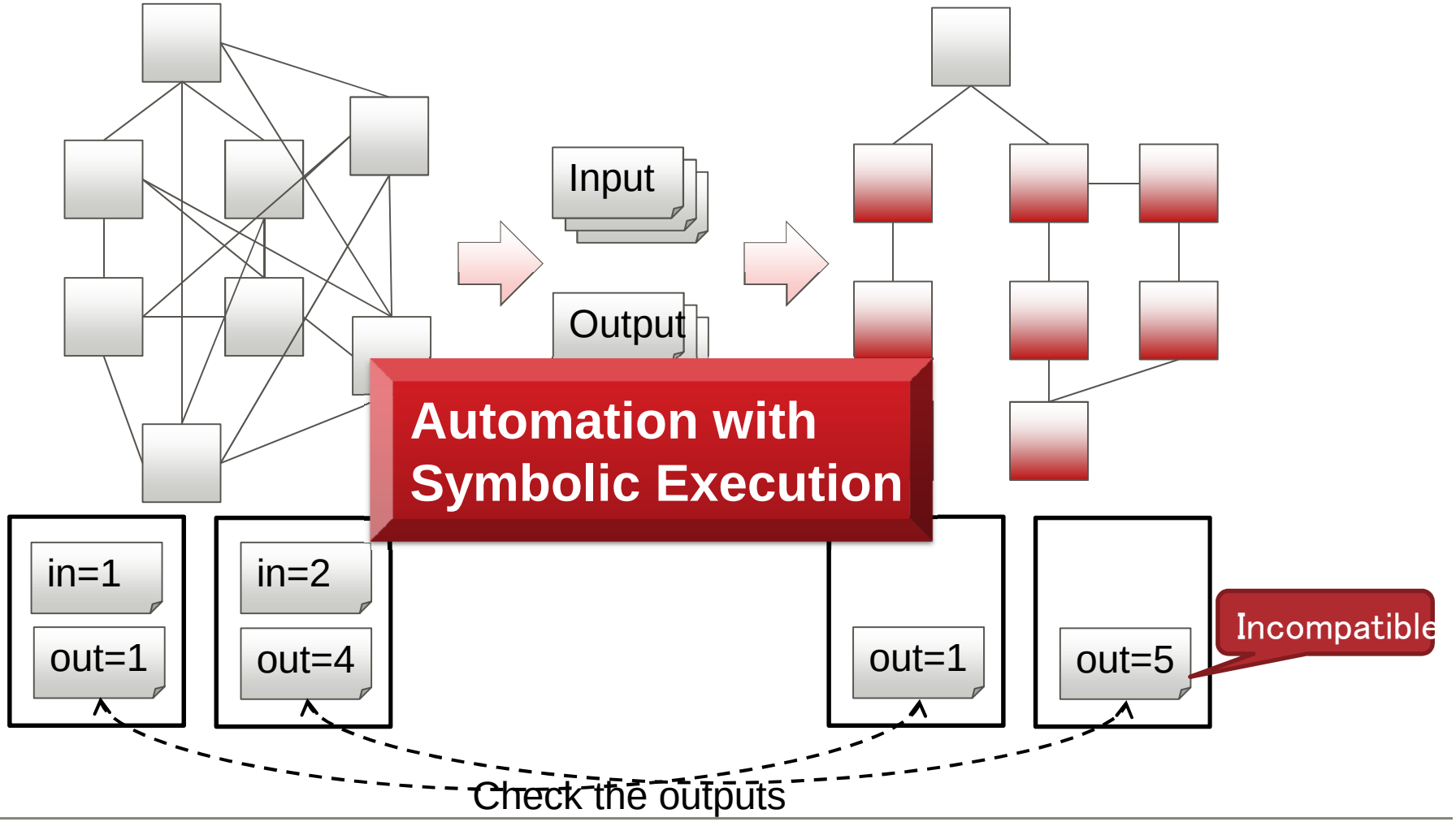


Main Issue:
Does the new system keep the same functionality of the old one?
⇒ Compatibility testing!

How to Test the Compatibility of the new System

FUJITSU

- Basic idea: Generate and run exhaustive test cases and record outputs on one system, then check the outputs with corresponding inputs on the other

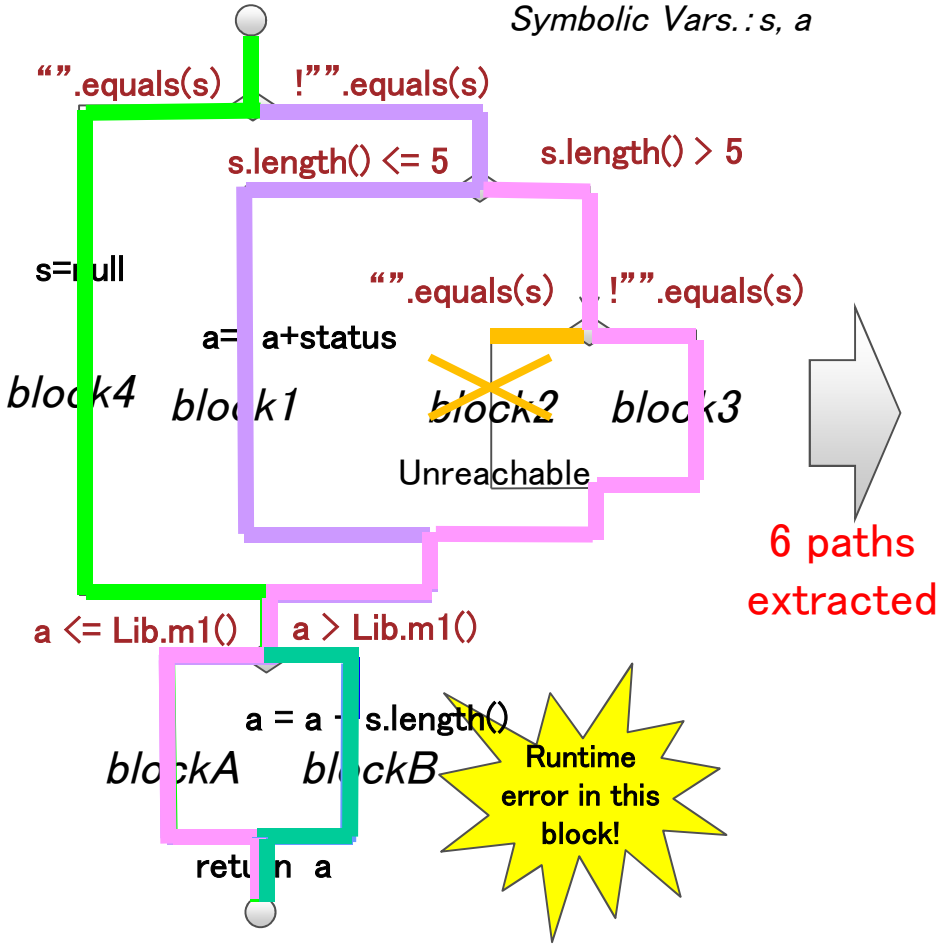


Test Generation through Symbolic Execution

FUJITSU

- Handle variables in target programs as **Symbolics** with constraints on its value, and obtain test data meeting the constraints

Flow for Program under Test



Tests to be Generated

Constraints to be met by variables

No	Test Data	Path Conditions
1	s = "", a = 0 Lib.m1() = 0	("".equals(s)) ∧ (a <= Lib.m1())
2	s = "", a = 0 Lib.m1() = -1	("".equals(s)) ∧ (a > Lib.m1())
3	s = " ", a = 0 status= 0 Lib.m1() = 0	(!"".equals(s)) ∧ (s.length() <= 5) ∧ (a+status<=Lib.m1())
4	s = " ", a = 1 status= 0 Lib.m1() = 0	(!"".equals(s)) ∧ (s.length() <= 5) ∧ (a + status>Lib.m1())
5	s=" "(6 whitespaces) a=0 Lib.m1()=0	(!"".equals(s)) ∧ (s.length() > 5) ∧ (a<=Lib.m1())
6	s=" "(6 whitespaces) a=0 Lib.m1()=1	(!"".equals(s)) ∧ (s.length() > 5) ∧ (a + status>Lib.m1())

Values meeting cond.

(*) Initial values are used for variables not referred in path conditions

Evaluation on a Re-engineering Project

FUJITSU

- Re-engineering of a SMTP library
  - As Is
    - The source code of the server products' monitor is different from that of the storage systems.
    - However their SMTP libraries have similar features
  - To Be
    - The both of SMTP libraries are unified
- Compatibility test Results

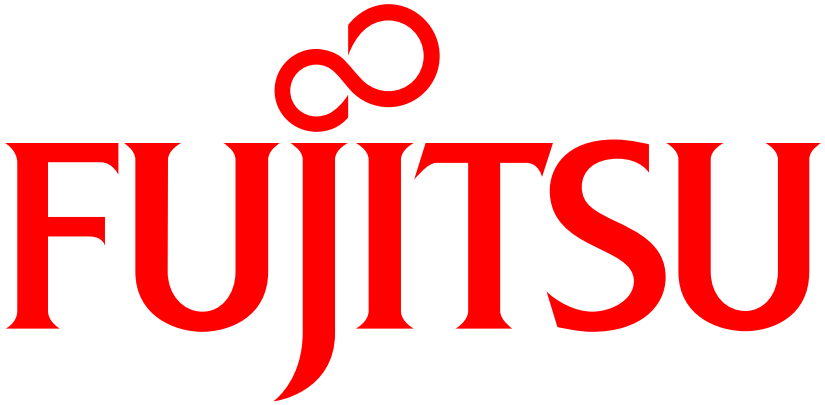
Comparison of Manual testing and our approach

	Manual testing	Our approach
Man-months	1.5	4
# of test cases	545	10846
# of detected bugs	27	27+5

Discussions

FUJITSU

- In addition to Understanding and Test, what are the steps requiring efforts during maintenance?
  - Automatic conversion of legacy code into higher level description etc.
- Efforts on earlier stages (better documents) will ease maintenance at later stages, but how can we motivate developers?
- Duration of software maintenance in general?
  - Which class of software should researchers target?
  - We are dealing with systems lived for 10+ years, but is it common?
- Are the two technologies introduced of *any relief at horizon*?



shaping tomorrow with you

Panel discussion

Lessons Learned on Software Maintenance: Any Relief at Horizon?

Roy Oberhauser
Aalen University
Germany

State of SW Maintenance

- What kind of SW maintenance is being done? [1]
  - Corrective – diagnosing and fixing (~20%)
  - Adaptive –coping with SW environment
  - Perfective – functional enhancements
  - Preventative – (4%)

Evolutionary development

- US SW industry employees 2010
  - 3M in SW maintenance, 800K in development (~80%) [2]

Proportionately maintenance is mostly about evolutionary development
- yet fixing defects seems our greatest concern

Maintenance Impacts and Importance

- Cost and criticality (especially infrastructure) to society & business
- Sheer code volume and defect rates
- Increased *value* of bugs/vulnerabilities
  - Greater usage and reliance on software systems
  - Increased *data* behind any breach
  - Increased *misuse market* for discovered defects
  - Easier widespread reuse/dispersment of defective code -> huge dependency chains (e.g., OpenSSL Heartbleed 1/2/…)

Correction work costs pale in relation to indirect costs and risks of a bug!

Potpourri of Trends Affecting Maintenance

- DevOps & Continuous Delivery -> Now a Continuum
- Changing public & business maintenance perception?
  - Hidden systems: PC-based vs. Cloud vs. Embedded
  - Bus slogan: "Leave the driving to us"…
  - Don't pay unless it hurts… Need forced "health insurance"?
    - Product backlog – what about a Maintenance backlog?
- Virtualization -> can isolate SW environment
  - Perhaps reduce adaptive maintenance?
- Forking OSS repositories -> Fix-It-Yourself
- Etc.

Some Maintenance Challenges

- Perfect implementation or perfect maintenance?
- API usage and semantics
- Software entropy and technical debt
- Agile software processes & generational comm.
  - Maintenance is typically a "step-child"
- Comprehending SoS impacts and interactions
  - Interdependencies across application boundaries
- But…
  - "Almost all grave software problems can be traced to conceptual mistakes made before programming started" -

- Prof. Jackson of MIT in Scientific American June 2006

Some Lessons Learned ?
Some Benefits Reaped?

- Our perceptions?
  - We all eat a healthy diet, right?
- Best wishes or best practices?
  - Execution of maintenance-relevant agile practices lag the rest
    - Refactoring, Test-driven development in the bottom 3 according to the Forrester Research Q3 2009 Global Agile Adoption Survey
  - Sprint Review of Bug Fixes?!!
- Lessons, well, it depends:
  - Organizational priorities, size, financing, cultural risk averseness
  - System criticality, etc.
- Human psychological influences not considered
  - Mood-aware programming/debugging [3]
  - Sleep & smart-phone distractions: driver crashes vs. programmers…
- One lesson "learned": Shared code transparency?

Forrester Research Q3 2009 Global Agile Adoption Survey

Supposed Relief on the Horizon?

- Software Maintenance Maturity Model (S3M)?
- Improved education, training, & certifications?
  - MOOCs and YouTube to the rescue?
- Sexy tools
  - Better analytical and design verification tools and metrics
    - Automated anomaly detection, debugging
    - Advances in formal verification
  - Automated bug repair or assistance
  - Software reverse engineering tools
- Millennials: Who cares about maintenance anyway?
  - Disposable Apps/Software? Dynamic Applications? End-User Programming?
  - Integrate "Digital Natives" into maintenance?

Conclusion

Since so much can go wrong…
No *one* technique or tool
can or will dominate SW maintenance,
it requires a *holistic human, social, and technical approach*

Best we can hope for…

- Increase awareness of *value* of maintenance
- *Incremental improvements* that slowly address a monumental amount of software already produced and to be maintained, and that which we are about to produce

Thank you!

References

[1] Eick, S., Graves, T., Karr, A., Marron, J., and Mockus, A. 2001. Does Code Decay? Assessing Evidence from Change Management Data. IEEE Transactions on Software Engineering. 27(1) 1-12.

[2] C. Jones. The Economics of Software Maintenance in the 21st Century, V3. 2006.

[3] Khan, I. A., Brinkman, W. P., & Hierons, R. M. (2011). Do moods affect programmers' debug performance?
Cognition, technology & work, 13(4), 245-258.

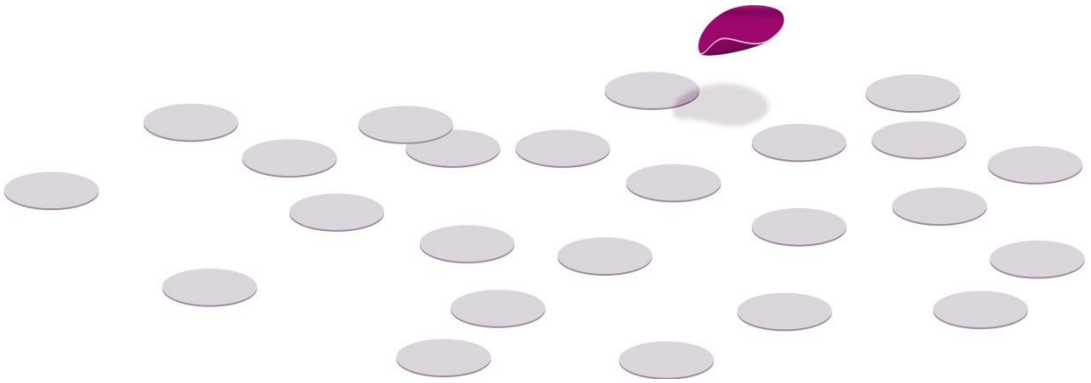
iteratec

KOMPETENZ, DIE ENTLASTET

Maintenance of Web Services

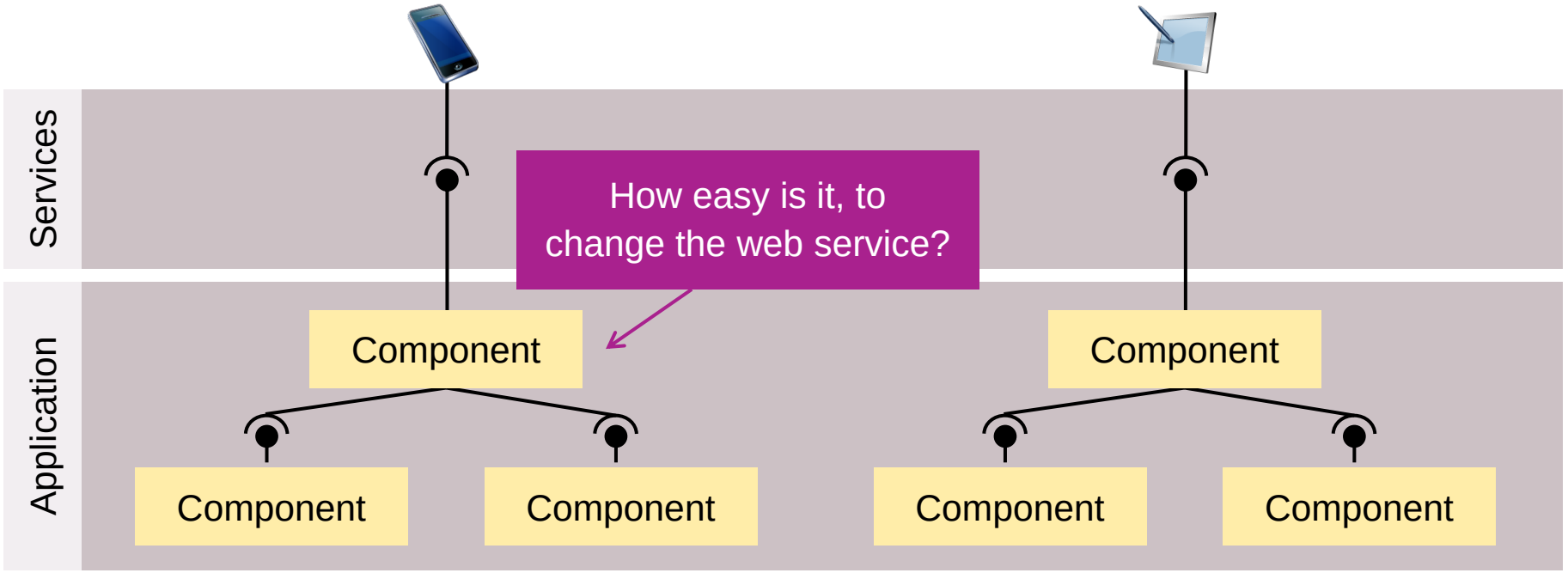
ICSEA 2014

Dr. Michael Gebhart



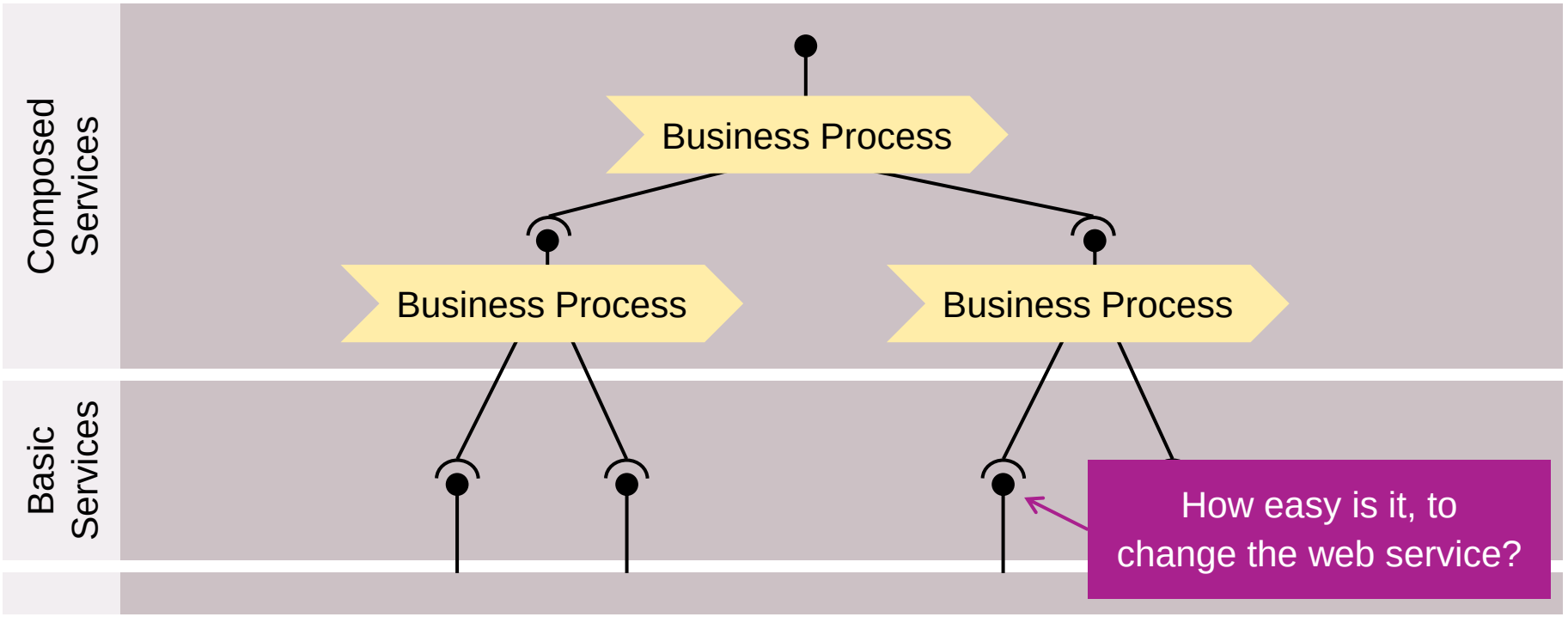
Maintenance of Web Services

Internal View



- Today, more and more web services are developed
  - e.g. RESTful web services as backend for apps on mobile devices
- Functionality to provide web services is part of the application
  - The quality of the entire system is strongly influenced by the quality of the web services
- More than ever, we need to design web services with care
- Maintenance with focus on the IT system

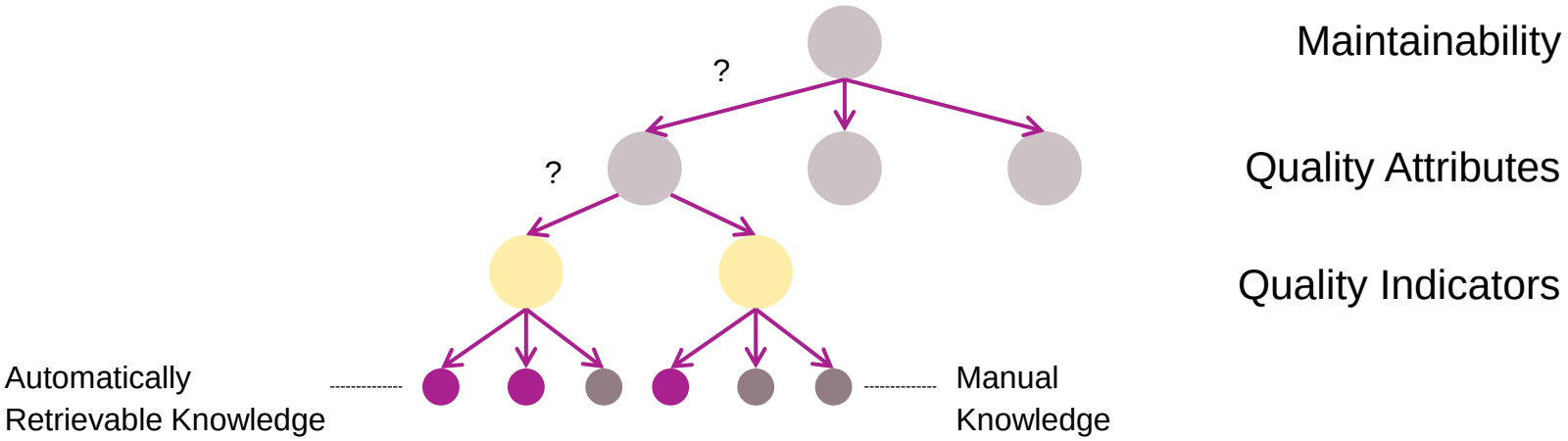
External View – Service-Oriented Architectures



- Services are understood as assets
- Quality characteristics that influence the maintainability: unique categorization (cohesion), loose coupling, autonomy, discoverability etc.
- Maintenance with focus on the service-oriented architecture

Creation of Maintainable Web Services

(Semi-)Automated Measurement of Quality Indicators



- Service-Oriented Architecture is business-driven
  - Often, necessary information is not part of the source code or interface description
  - Manual information is necessary
- Creation of a quality model with best practices as quality indicators that refer to web services as artifacts
- Combination with manual knowledge
  - Interaction with experts is necessary
  - Hybrid approach is proposed that combines automated analysis with manual knowledge

Recommended Literature

Quality Analysis of Services and Service-Oriented Architectures

- Gebhart, M., Giessler, P., Burkhardt, P., & Abeck, S. (2014). Quality-Oriented Requirements Engineering for Agile Development of RESTful Participation Service. In H. Mannaert, L. Lavazza, R. Oberhauser, M. Kajko-Mattsson, & M. Gebhart (Eds.), *Proceedings of the Ninth International Conference on Software Engineering Advances (ICSEA) 2014* (pp. 69-74). ISBN: 978-1-61208-367-4.
- Gebhart, M. (2014). Query-Based Static Analysis of Web Services in Service-Oriented Architectures. *International Journal on Advances in Software, 7*(1&2), 136-147.
- Gebhart, M., & Sejdovic, S. (2012). Quality-Oriented Design of Software Services in Geographical Information Systems. *International Journal on Advances in Software, 5*(3&4), 293-307. Gebhart, M. (2012). Service Identification and Specification with SoaML. In A. D. Ionita, M. Litoiu, & G. Lewis (Eds.), *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments* (pp. 102-125). Hershey, PA: IGI Global. doi: 10.4018/978-1-4666-2488-7. ISBN: 978-1-46662488-7.
- Gebhart, M., Baumgartner, M., & Abeck, S. (2010). Supporting Service Design Decisions. In J. Hall, H. Kaindl, L. Lavazza, G. Buchgeher, & O. Takaki (Eds.), *Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA) 2010* (pp. 76-81). doi: 10.1109/ICSEA.2010.19
- Gebhart, M., Baumgartner, M., Oehlert, S., Blersch, M., & Abeck, S. (2010). Evaluation of Service Designs based on SoaML. In J. Hall, H. Kaindl, L. Lavazza, G. Buchgeher, & O. Takaki (Eds.), *Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA) 2010* (pp. 7-13). doi: 10.1109/ICSEA.2010.8
- Gebhart, M., & Abeck, S. (2009). Rule-Based Service Modeling. In K. Boness, J. M. Fernandes, J. G. Hall, R. J. Machado, & R. Oberhauser (Eds.), *Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA) 2009* (pp. 271-276). doi: 10.1109/ICSEA.2009.48

iteratec

KOMPETENZ, DIE ENTLASTET

Thank you for your attention

Dr. Michael Gebhart
michael.gebhart@iteratec.de