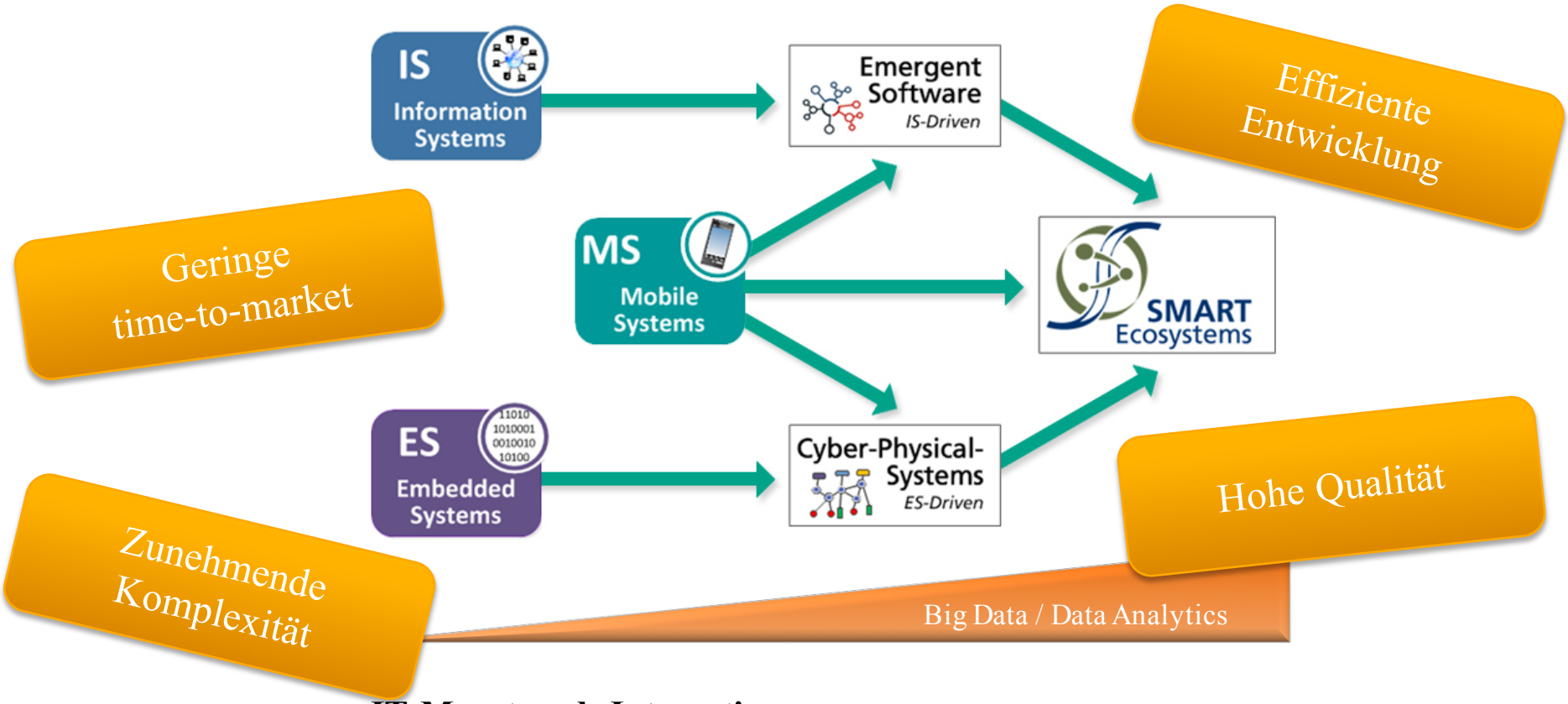
Fraunhofer IESE

Nutzung von Architekturinformationen zur Beherrschung der Komplexität im Software-Integrationstest



Dr. Frank Elberzhager

15. Oktober 2015
TAV, Paderborn



IT-Megatrend: Integration

Stärkere Planung, Steuerung, und Durchführung von Integrationstests basierend auf Architekturinformationen.

© Fraunhofer IESE



© Fraunhofer IESE

In summary, observations 1–4 are advantages of incremental testing, and observations 5–6 are disadvantages. Given current trends in the computing industry (hardware costs are decreasing, and labor costs and the consequences of software errors are increasing), and given the fact that the earlier an error is found, the lower the cost of repairing it, one sees that observations 1–4 are increasing in importance while observation 5 is becoming less important. Observation 6 seems to be a weak disadvantage, if one at all. This leads to the conclusion that incremental testing is superior.

TOP-DOWN VERSUS BOTTOM-UP TESTING

Given the conclusion of the previous section, that incremental testing is superior to nonincremental testing, two incremental strategies are explored: *top-down* and *bottom-up* testing. Before discussing them, however, several misconceptions should be clarified. First, the terms "top-down testing," "top-down development," and "top-down design" are often used as synonyms. Top-down testing and top-down development are synonyms (they represent a strategy of ordering the coding and testing of modules), but top-down design is something quite different and independent. A program that was designed in a top-down fashion can be incrementally tested in either a top-down or a bottom-up fashion.

Second, bottom-up testing (or bottom-up development) is often mistakenly equated with nonincremental testing. The reason is that bottom-up testing begins in a manner that is identical to a nonincremental test (i.e., when the bottom, or terminal, modules are tested), but as we saw in the previous section, bottom-up testing is an incremental strategy. Finally, since both strategies are incremental, the advantages of incremental testing are not repeated here; only the differences between top-down and bottom-up testing are discussed.

Top-Down Testing

The top-down strategy starts with the top, or initial, module in the program. After this, there is no single "right" procedure for selecting the next module to be incrementally tested; the only rule is that to be eligible to be the next module, at least one of the

module's superordinate (calling) modules must have been tested previously.

Figure 5.8 is used to illustrate the strategy. A–L are the 12 modules in the program. Assume that module J contains the program's I/O read operations and module I contains the write operations.

The first step is the testing of module A. To accomplish this, stub modules representing B, C, and D must be written. Unfortunately, the production of stub modules is often misunderstood; as evidence, one often sees such statements as "a stub module need only write a message stating 'we got this far,'" and "in many cases, the dummy module [stub] simply exits—without doing any work at all." In most situations, these statements are false. Since module A calls module B, A is expecting B to perform some work; this work most likely is some result (e.g., output arguments) returned to A. If the stub simply returns control or writes an error message without returning a meaningful result, module A will fail, not because of an error in A, but because of a failure of the stub to simulate the corresponding module. Moreover, returning a "wired-in" output from a stub module is often insufficient. For instance, consider the task of writing a stub representing a square-root routine, a table-search routine, an "obtain corresponding master-file record" routine, and the like. If the stub returns a fixed wired-in output, but not having the particular value expected by the calling module during this invocation, the

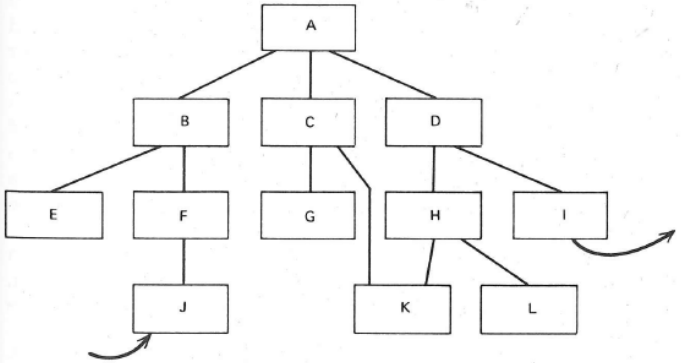
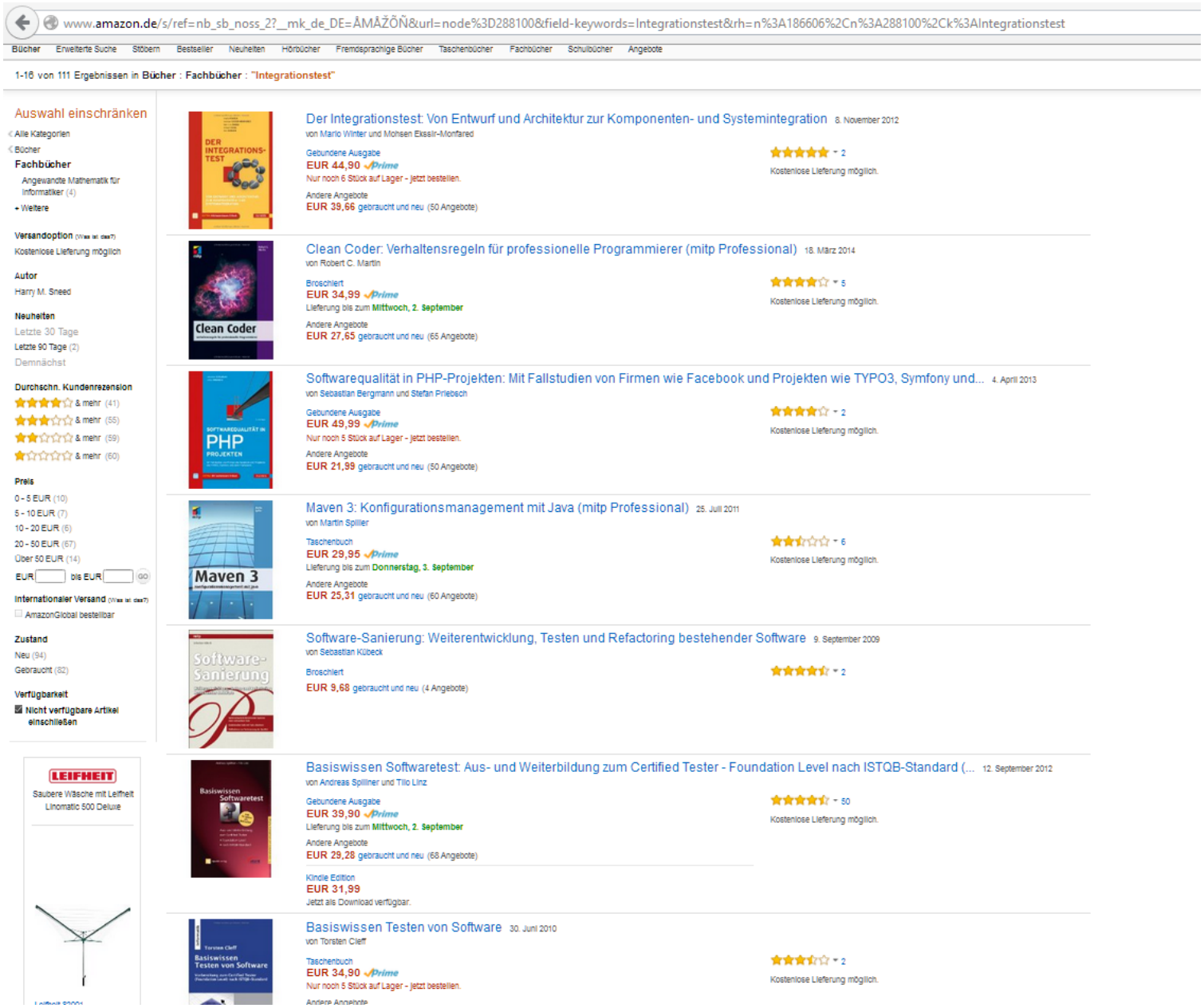


Figure 5.8 **Sample 12-module program.**

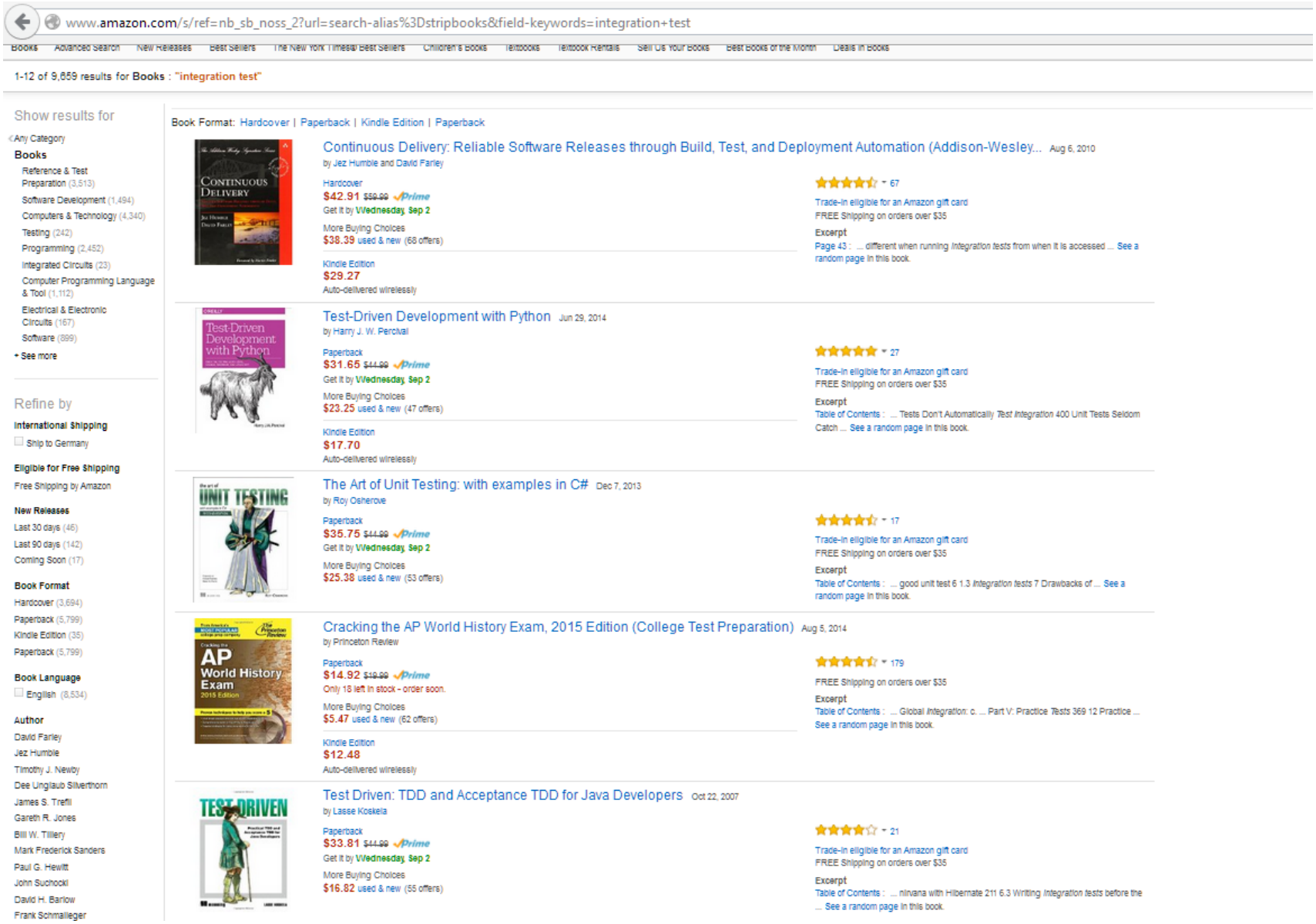
© Fraunhofer IESE

Fraunhofer IESE



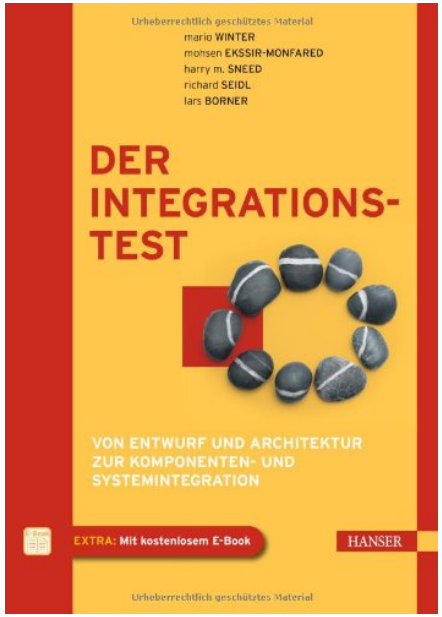
© Fraunhofer IESE

Fraunhofer IESE



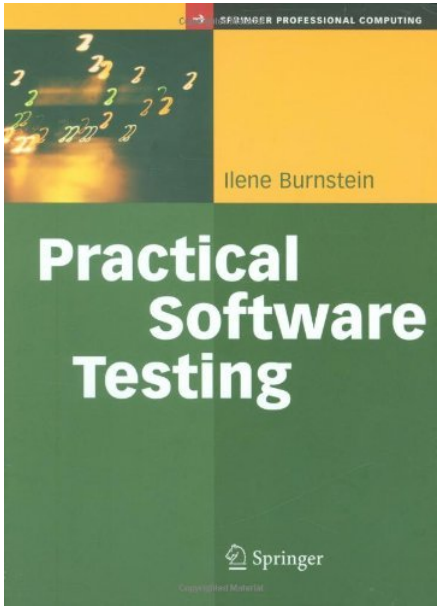
© Fraunhofer IESE

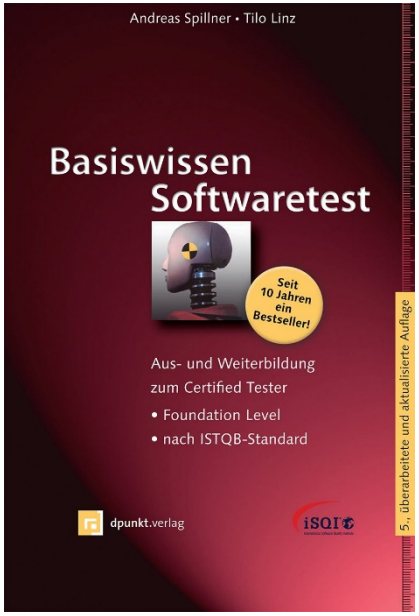
Fraunhofer IESE











© Fraunhofer IESE

Fraunhofer IESE

https://insights.sei.cmu.edu/sei_blog/2011/08/improving-testing-outcomes-through-software-architecture.html

CMU SEI CERT Division Digital Library SEI Insights

Software Engineering Institute
Carnegie Mellon University

SEI Insights

Home Blogs Authors Search Search

Home > SEI Blog > Improving Testing Outcomes Through Software Architecture

SEI Blog

The Latest Research in Software Engineering and Cybersecurity

Improving Testing Outcomes Through Software Architecture

POSTED ON AUGUST 22, 2011 BY PAUL CLEMENTS IN ARCHITECTURE

Testing plays a critical role in the development of software-reliant systems. Even with the most diligent efforts of requirements engineers, designers, and programmers, faults inevitably occur. These faults are most commonly discovered and removed by testing the system and comparing what it does to what it is supposed to do. This blog posting summarizes a method that improves testing outcomes (including efficacy and cost) in a software-reliant system by using an architectural design approach, which describes a coherent set of architectural decisions taken by architects to help meet the behavioral and quality attribute requirements of systems being developed.

Developers of software-reliant systems must address several testing-related challenges. For example, testing is expensive and can account for more than 50 percent of a project's schedule and budget, depending on the criticality of the system. Unfortunately, some organizations assign a budget for testing and stop when that budget is consumed. Safety-critical software organizations also have a budget, but they often must reach a confidence level independent of the expenditures to meet certification standards. In either situation, improving the efficacy and cost of testing is essential to meeting requirements and business goals.

Another challenge is that few organizations inform the testing process by considering the software architecture, which comprises the structure of the software elements in a system, the externally visible properties of those elements, and the relationships among them. Ignoring or

© Fraunhofer IESE

Fraunhofer IESE



Integrationstest wird betrachtet, aber Unterstützung bei der Umsetzung ist häufig begrenzt.

© Fraunhofer IESE

Integration

IEEE Standard Glossary: “The process of combining software components, hardware components, or both into an overall system.“

Test

IEEE 829: “Activity in which a system or component is executed under specified conditions, the results are observed and recorded, and an evaluation is made of some aspect of the system or component.”

Integrationstest

IEEE Standard Glossary: “Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction among them.“

Integrationstest

“Im Integrationstest werden die Abhängigkeiten und das Zusammenspiel zwischen einzelnen Bausteinen eines Software-Systems getestet.”

[M. Winter, M. Ekssir-Monfared, H. Sneed, R. Seidl, L. Borner, Der Integrationstest]

© Fraunhofer IESE

Architektur

The **structure or structures** of the **system**, which comprise software **elements**, the **externally visible properties** of those elements, and the **relationships** among them.

[Software Architecture in Practice, L. Bass, P. Clements, R. Kazman]

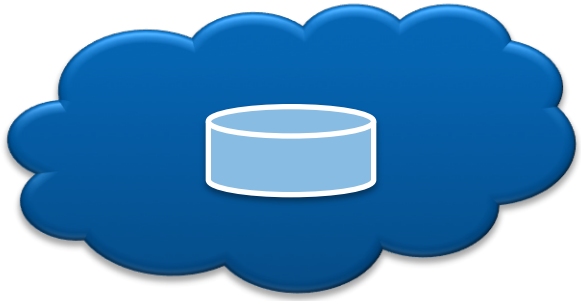
The set of **principal design decisions** made about the **system**.

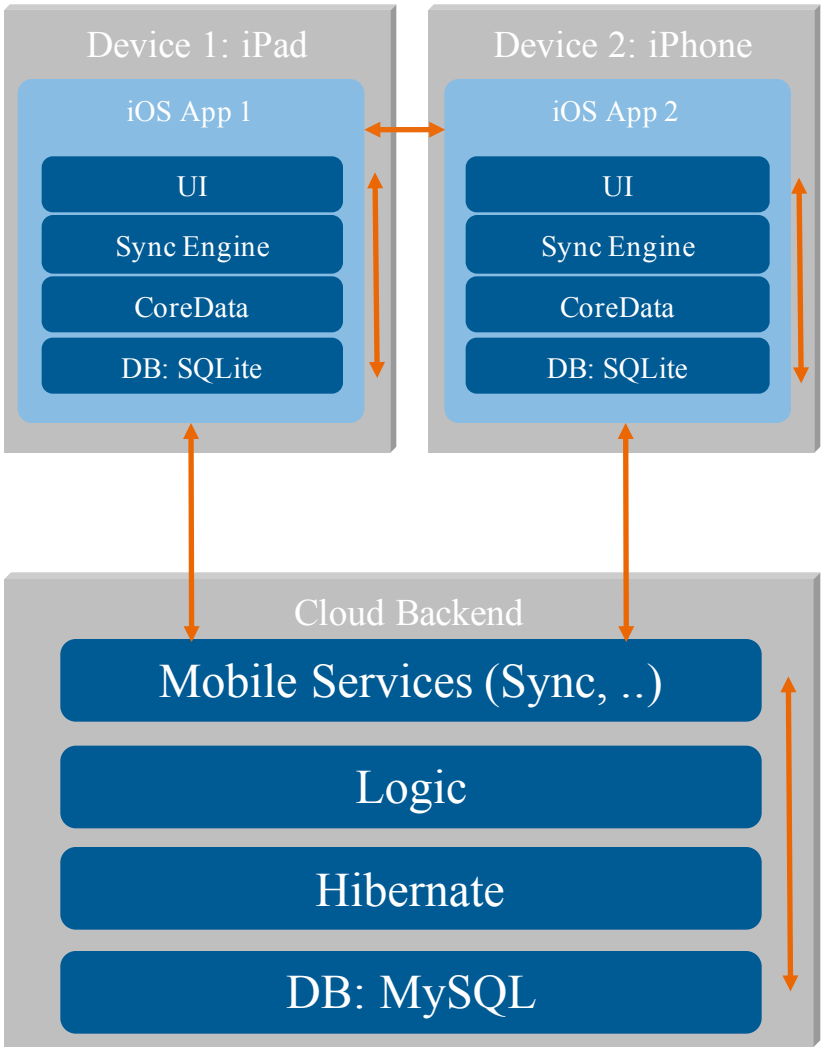
[Software Architecture: Foundations, Theory, and Practice, R.N. Taylor, N. Medvidovic, E.M. Dashofy]

© Fraunhofer IESE

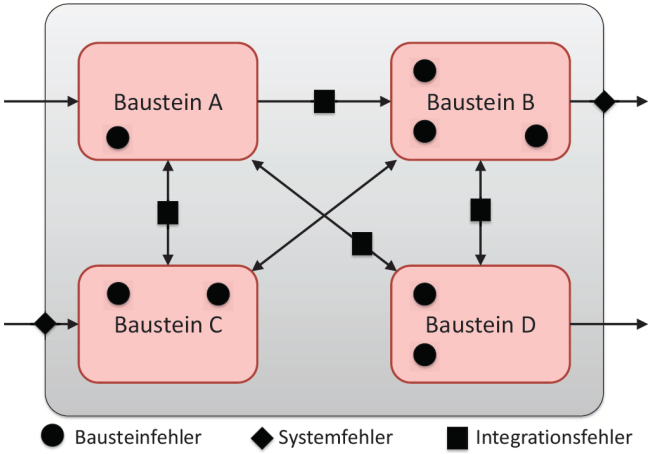








© Fraunhofer IESE





Integrationstest wird oftmals nicht als dedizierte Phase oder Aktivität durchgeführt.

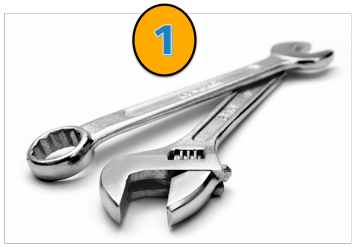
© Fraunhofer IESE

Best Practices & Konzepte



Enge und frühe Zusammenarbeit zwischen Architekten, Testern, und Projektleiter.

© Fraunhofer IESE



Enge und frühe Zusammenarbeit zwischen Architekten, Testern, und Projektleiter.



Mike Manager

. . .

Teststrategie
Ressourcenplanung



Arnold Architekt

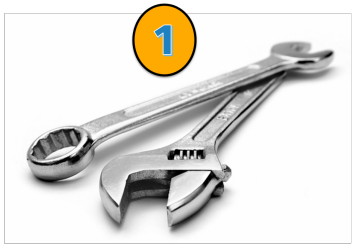
Austausch
Architekturinformationen
& Testbarkeit



Tanja Tester

© Fraunhofer IESE

Fraunhofer IESE



Enge und frühe Zusammenarbeit zwischen Architekten, Testern, und Projektleiter.

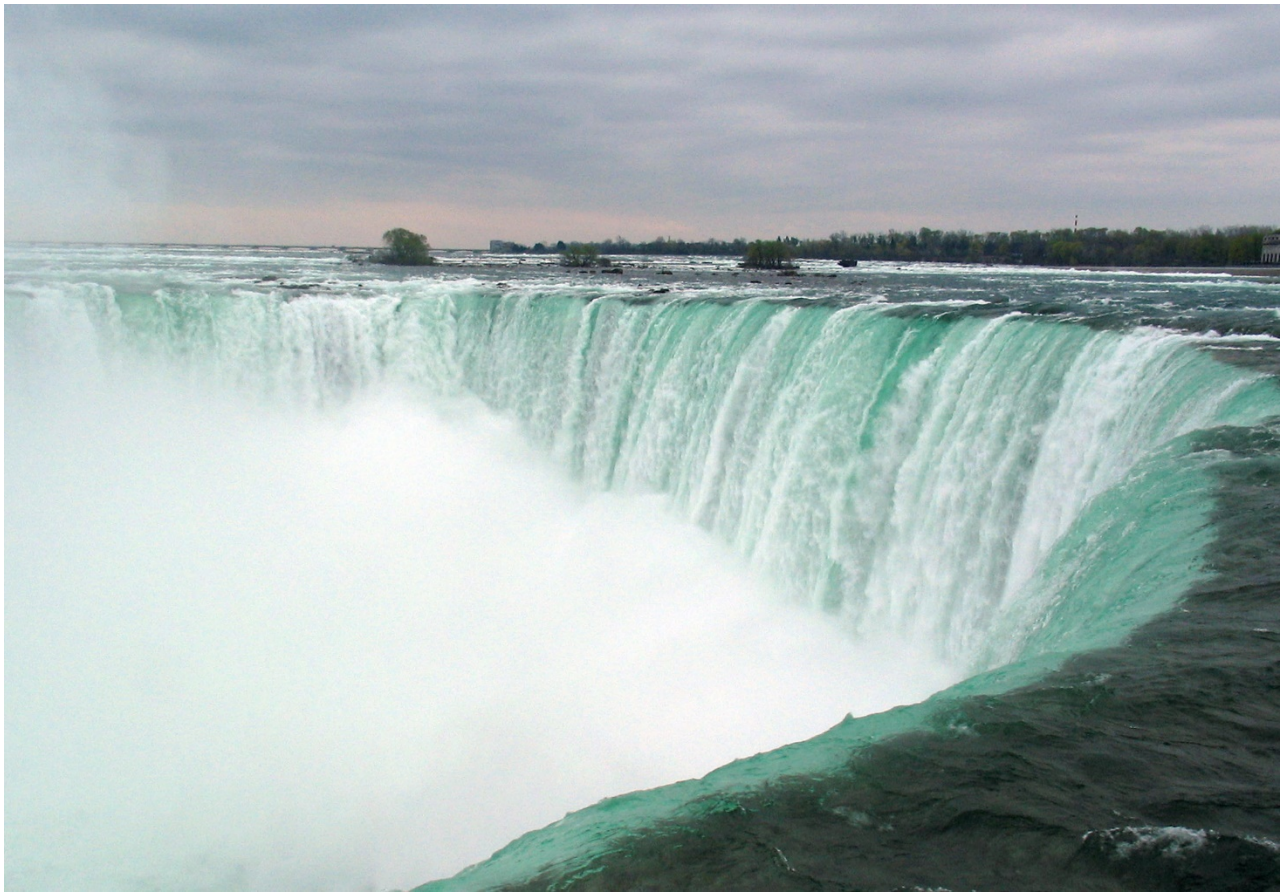


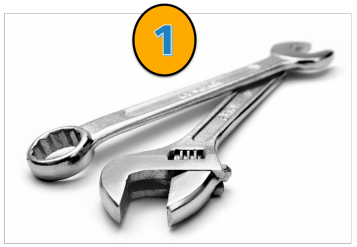
Kleine Teams mit wenig Zeit, wenig definierte und gelebte Prozesse, kaum Dokumente

Integrationstest wird „irgendwie“ mitgemacht

© Fraunhofer IESE

Fraunhofer IESE





Enge und frühe Zusammenarbeit zwischen Architekten, Testern, und Projektleiter.

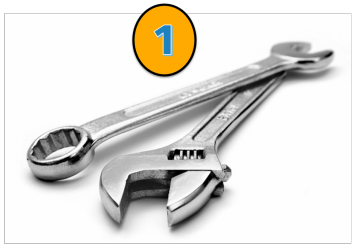
Große Abteilungen mit Silodenken, riesige Dokumente (die keiner liest), extreme Systemkomplexität

Integrationstest wird ineffektiv und ineffizient mitgemacht

© Fraunhofer IESE

Fraunhofer IESE





Enge und frühe Zusammenarbeit zwischen Architekten, Testern, und Projektleiter.

Keine dedizierten Rollen, wenig Dokumentation

Integrationstest wird erfahrungsbasiert mitgemacht

© Fraunhofer IESE

Fraunhofer IESE



Enge und frühe Zusammenarbeit zwischen Architekten, Testern, und Projektleiter.

Sollte klar sein, wird aber oftmals nicht (gänzlich) gelebt.

© Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams).

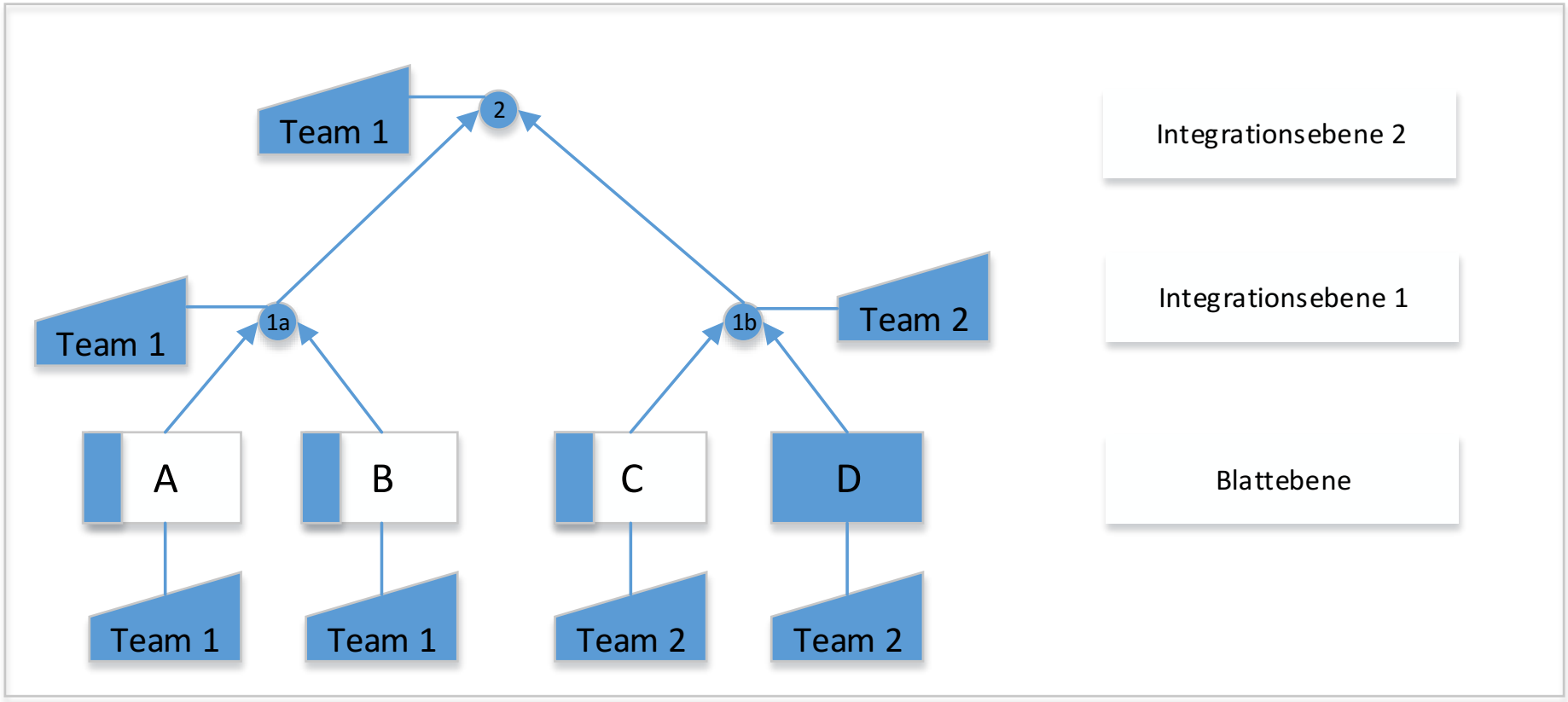
© Fraunhofer IESE



Tanja Tester



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams)



© Fraunhofer IESE

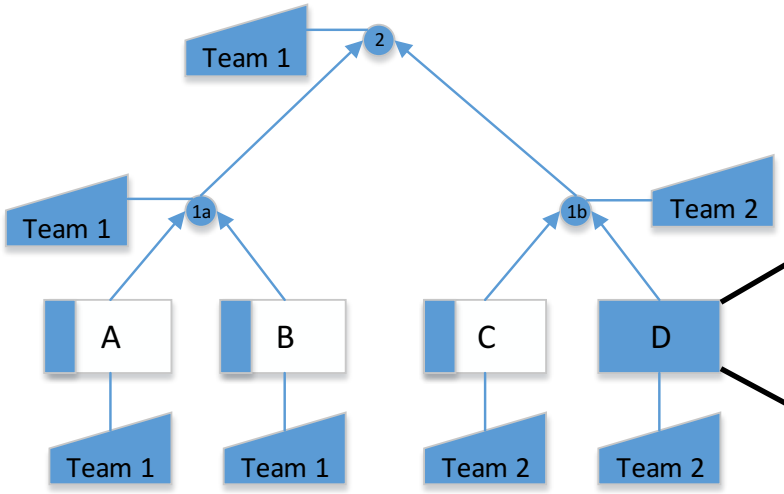
Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams)



Tanja Tester



Codeklasse
Menge von Codeklassen
Cluster
Konnektor
Datenbank
GUI Komponente
…

© Fraunhofer IESE

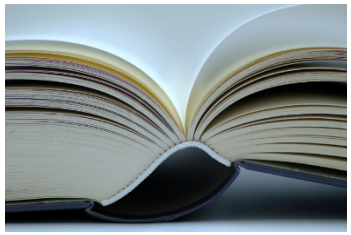
Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams)

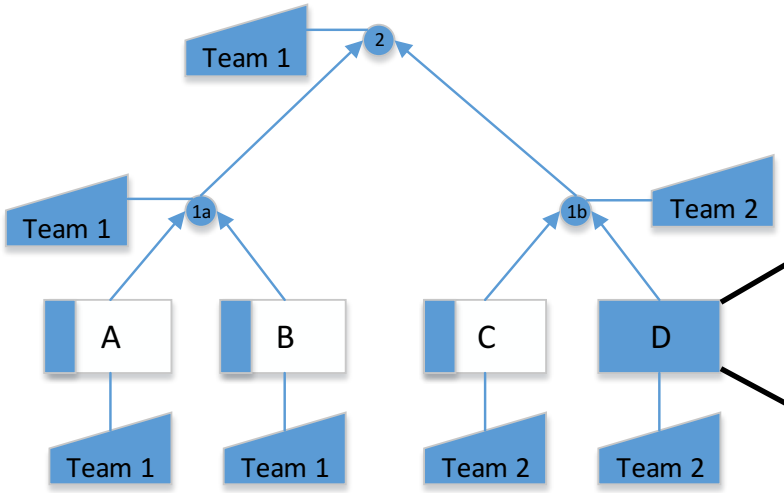


Tanja Tester





Arnold Architekt



Codeklasse
Menge von Codeklassen
Cluster
Konnektor
Datenbank
GUI Komponente
…

© Fraunhofer IESE

Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams).

Basis für den Tester zur frühen, vollständigen sowie vereinfachten Übersicht der Struktur (hilfreicher, je komplexer die Architektur).

© Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams).

Forcierung der frühen Kommunikation zwischen Architekt und Tester.

© Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams).

Grundlage für weitere Entscheidungen, z.B. Umfang des Integrationstests, umzusetzender Automatisierungsgrad.

© Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams).

Ebenfalls frühe Rückmeldung bzgl. der Testbarkeit an den Architekten möglich.

© Fraunhofer IESE



Der **Integrationsbaum** liefert eine einfache Unterstützung (Übersicht der Architekturelemente und der Teams).

Vereinfachte Planung der Integrationsreihenfolge unter Berücksichtigung mehrerer Teams.

© Fraunhofer IESE



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

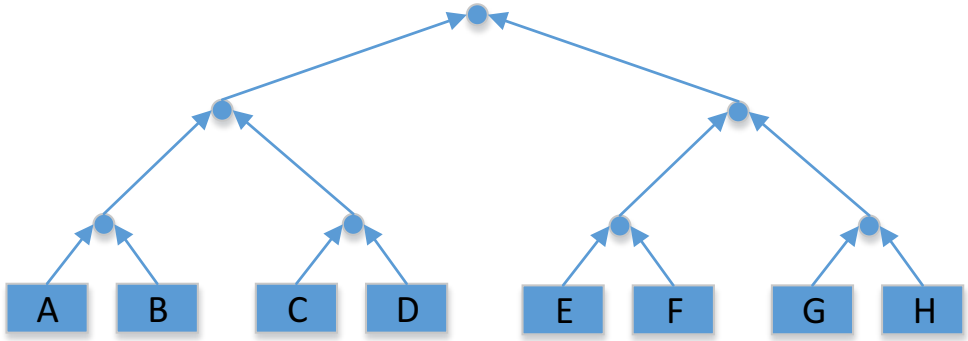
© Fraunhofer IESE

Vereinfachtes Beispiel (Konzept)



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Integrationsbaum



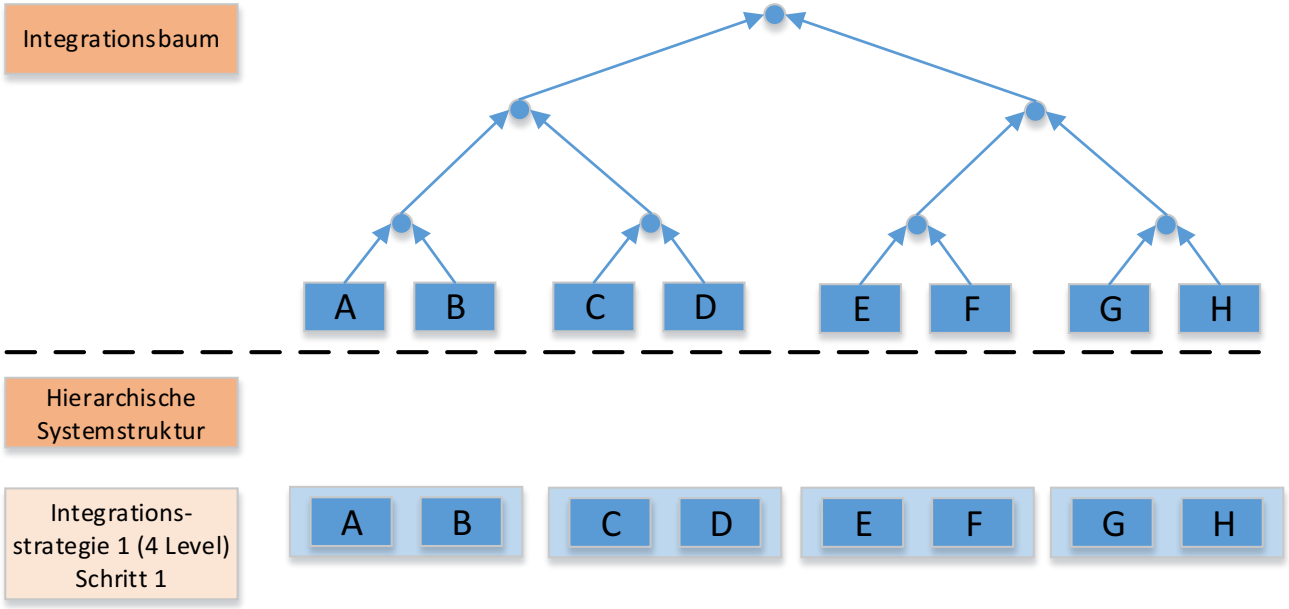
© Fraunhofer IESE

Fraunhofer IESE

Vereinfachtes Beispiel (Konzept)



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).



© Fraunhofer IESE

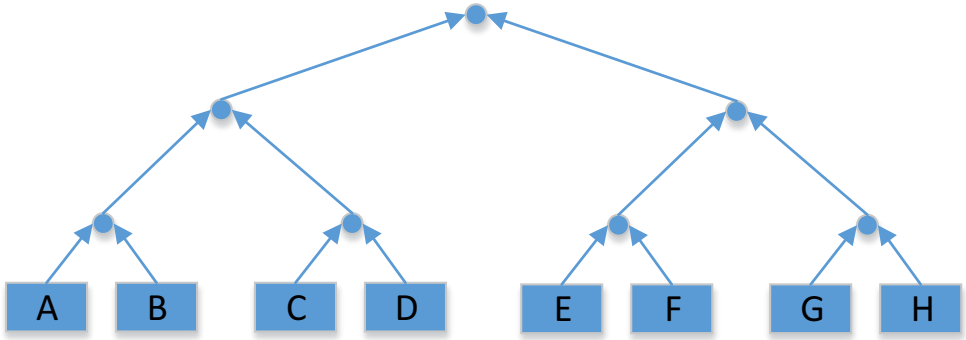
Fraunhofer IESE

Vereinfachtes Beispiel (Konzept)



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Integrationsbaum



Hierarchische Systemstruktur

Integrations-strategie 1 (4 Level) Schritt 2



© Fraunhofer IESE

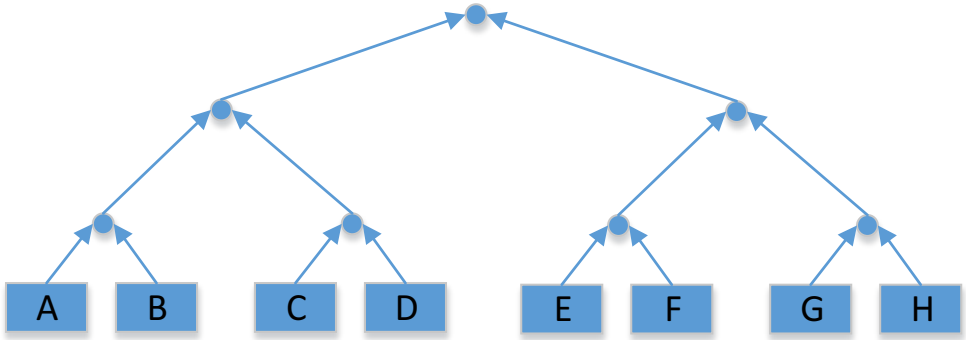
Fraunhofer IESE

Vereinfachtes Beispiel (Konzept)



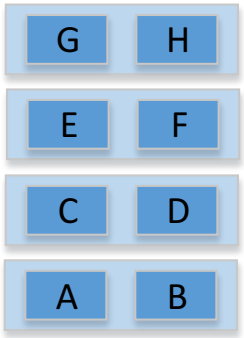
Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Integrationsbaum



Hierarchische Systemstruktur

Integrations-strategie 1 (4 Level) Schritt 3



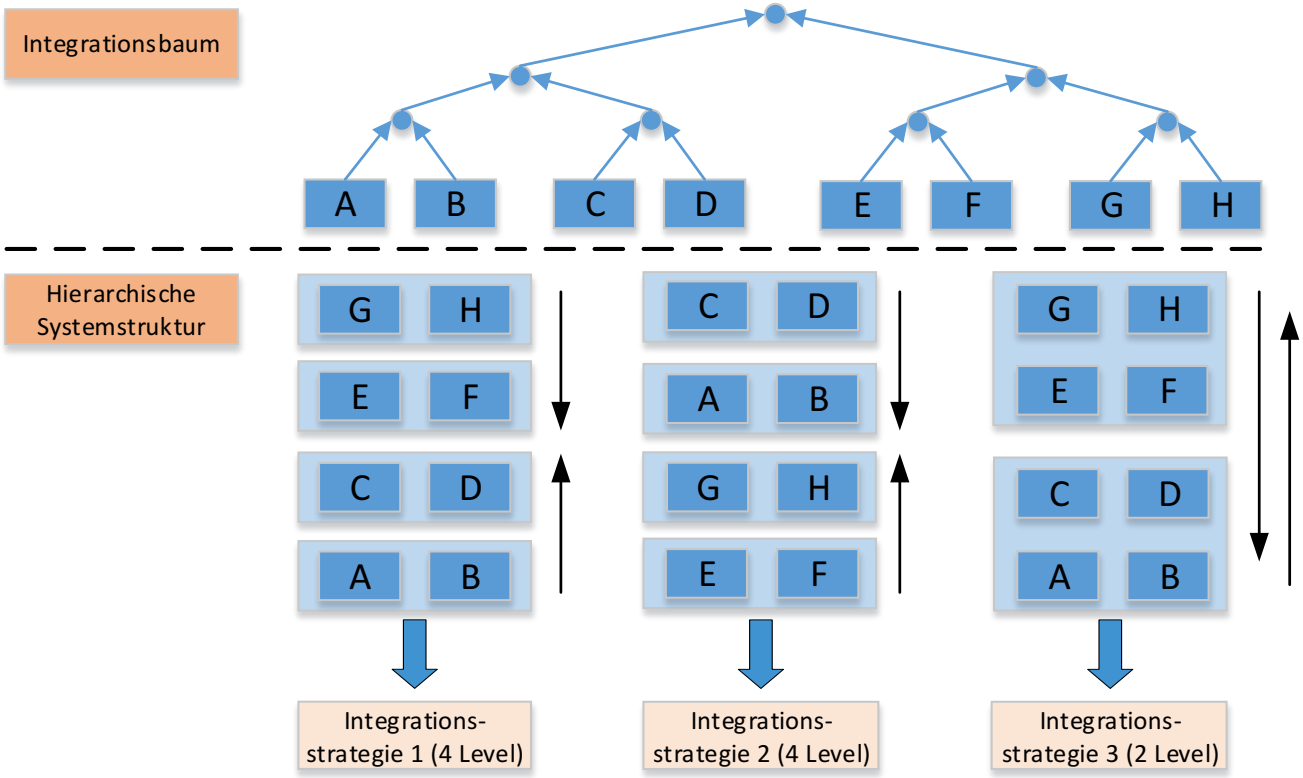
© Fraunhofer IESE

Fraunhofer IESE

Vereinfachtes Beispiel (Konzept)



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).



Ableitung der konkreten Strategie nur mit Hilfe von Architekturinformationen möglich.

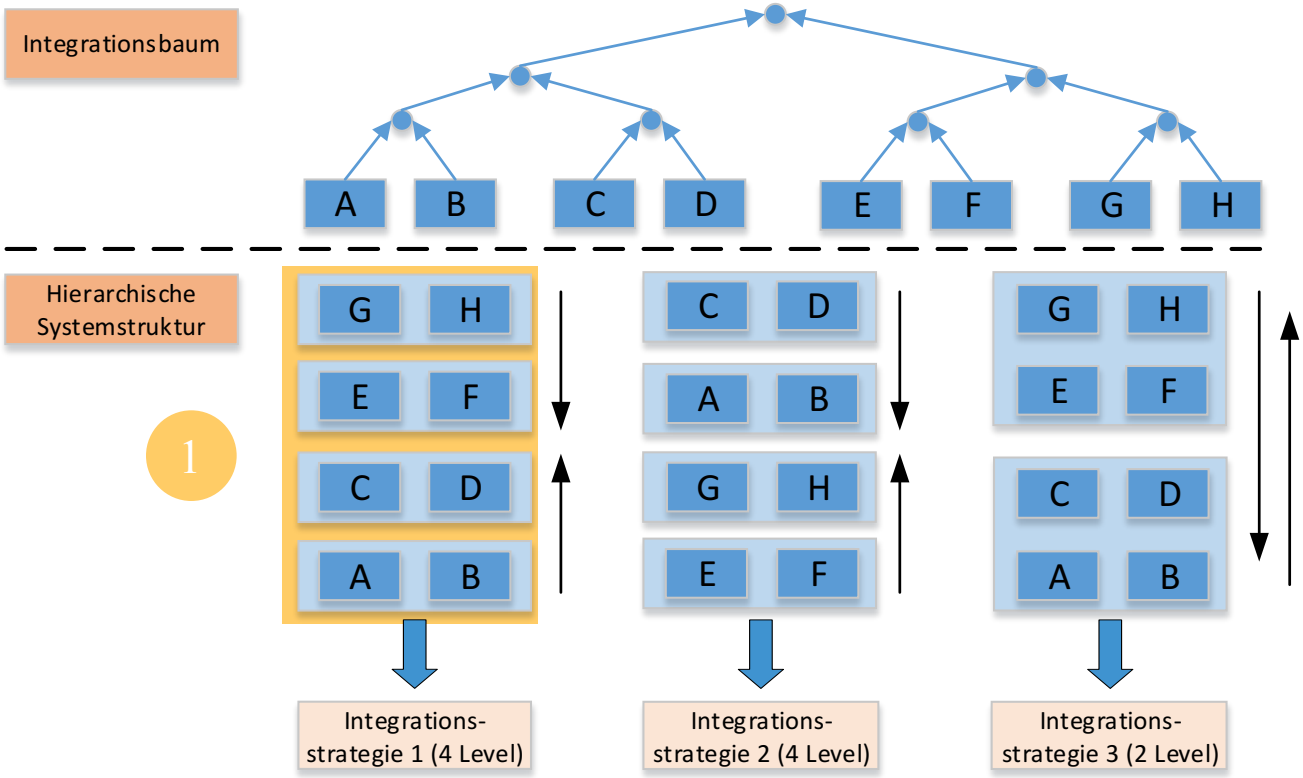
© Fraunhofer IESE

Fraunhofer IESE

Vereinfachtes Beispiel (Vorgehen)



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).



Ableitung der konkreten Strategie nur mit Hilfe von Architekturinformationen möglich.

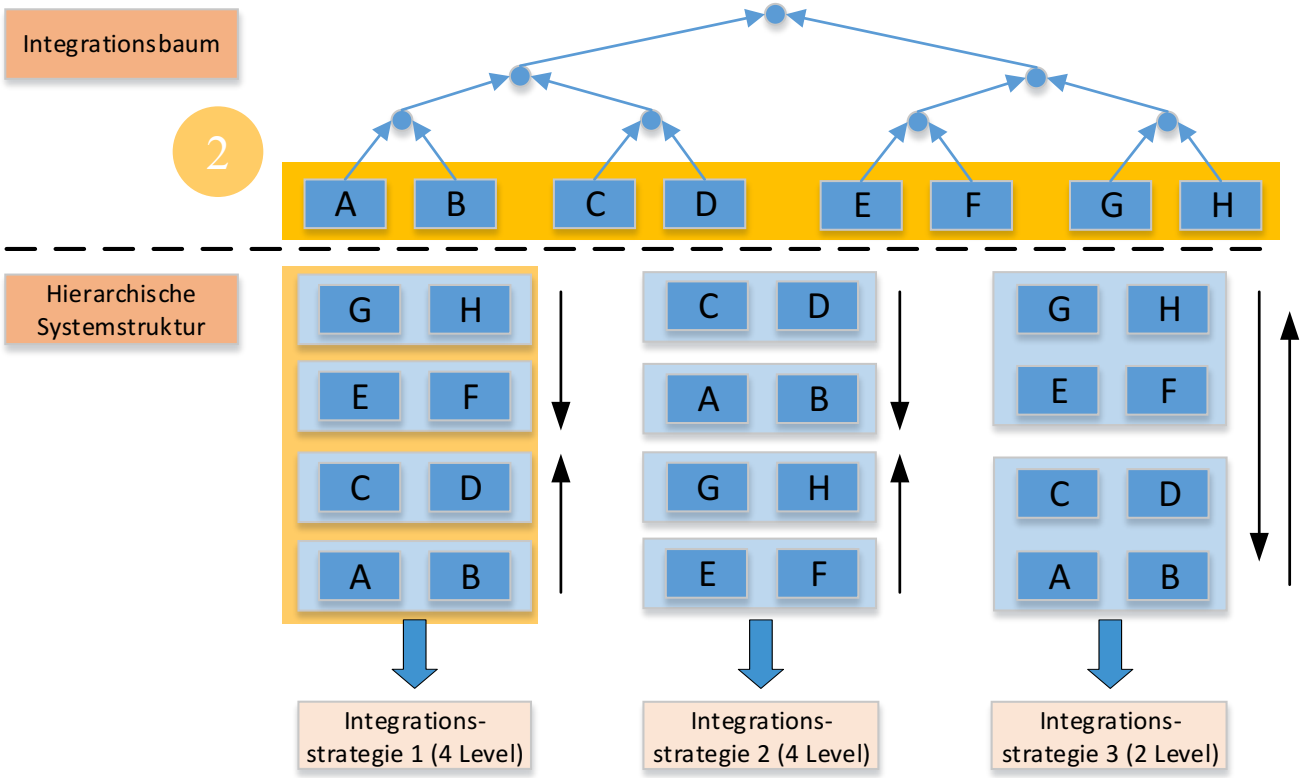
© Fraunhofer IESE

Fraunhofer IESE

Vereinfachtes Beispiel (Vorgehen)



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).



Ableitung der konkreten Strategie nur mit Hilfe von Architekturinformationen möglich.

© Fraunhofer IESE

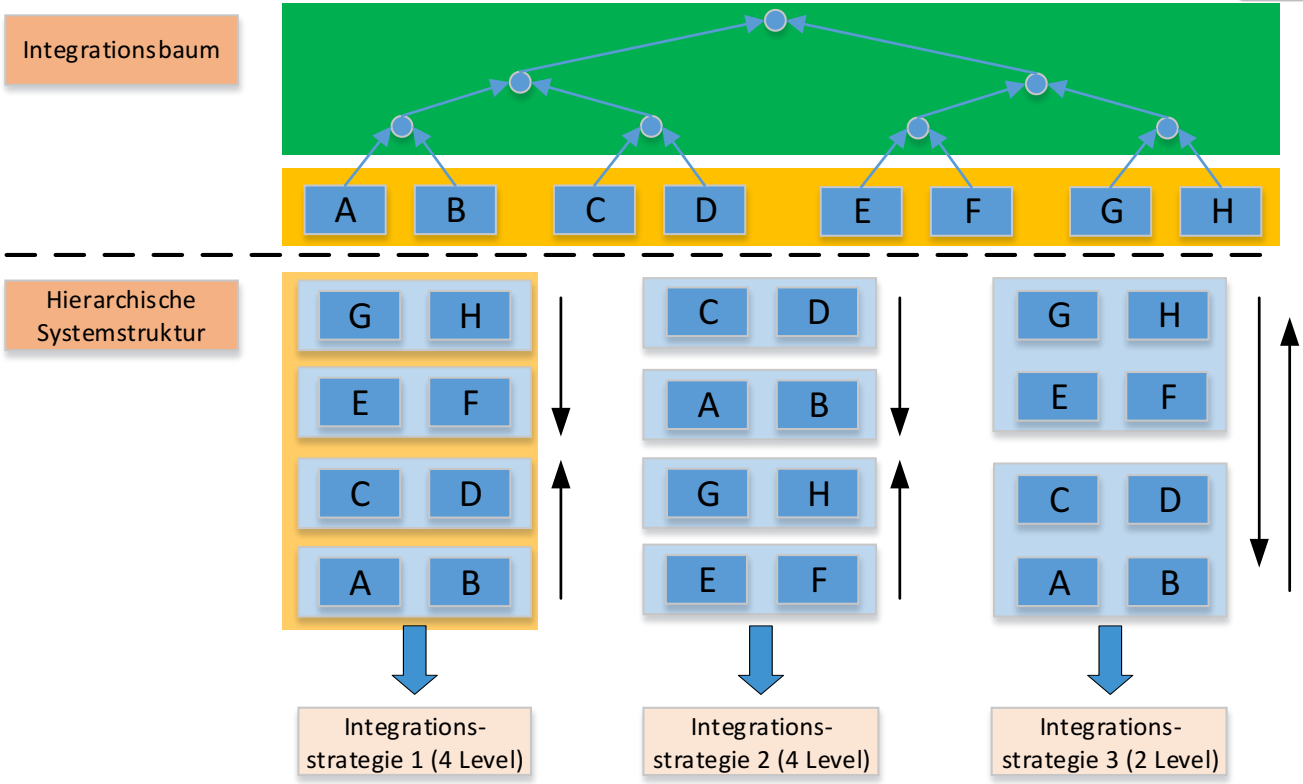
Fraunhofer IESE

Vereinfachtes Beispiel (Vorgehen)



3

Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).



Ableitung der konkreten Strategie nur mit Hilfe von Architekturinformationen möglich.

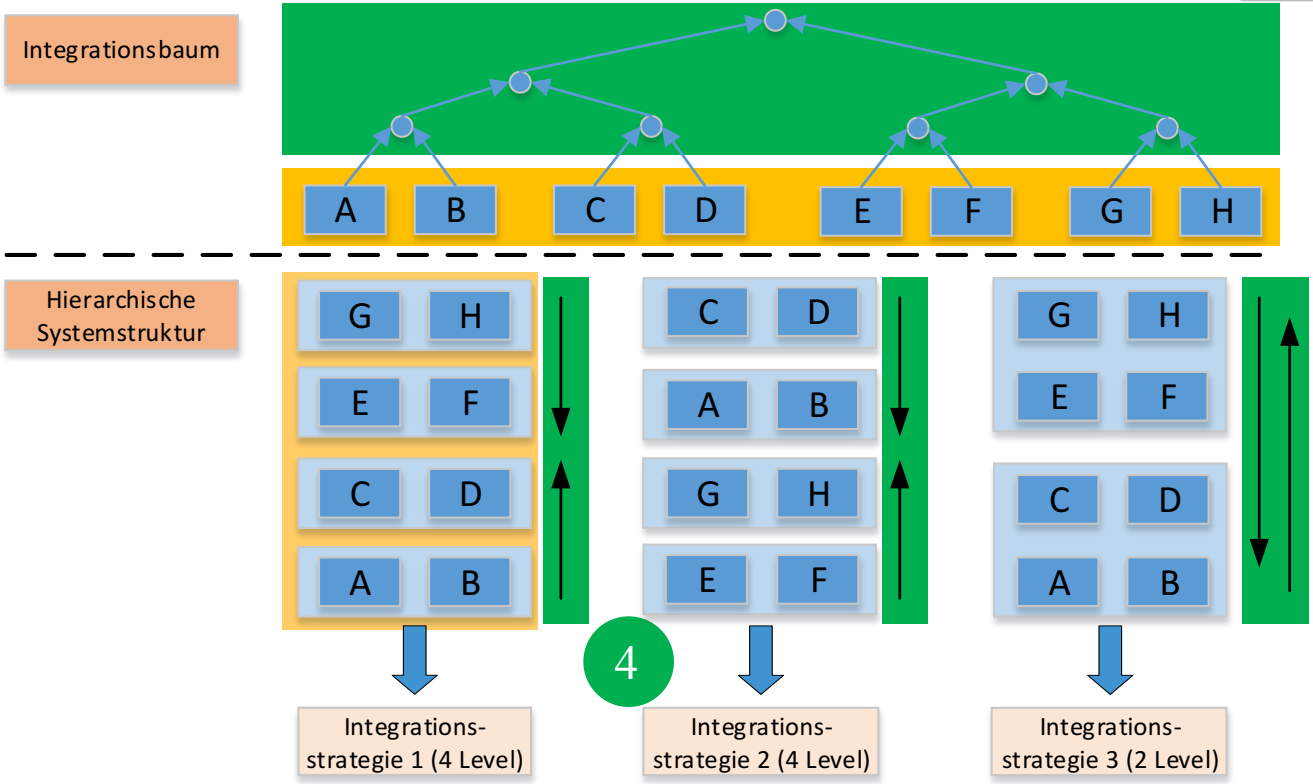
© Fraunhofer IESE

Fraunhofer IESE

Vereinfachtes Beispiel (Vorgehen)



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).



Ableitung einer konkreten Integrationsteststrategie (z.B. bottom-up, top-down) hängt oftmals von weiteren Faktoren ab (z.B. funktionaler Schnitt, Ressourcen, Entwicklungsreihenfolge).

© Fraunhofer IESE

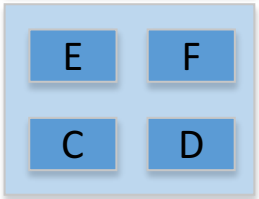
Vereinfachtes Beispiel



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Hierarchische Systemstruktur



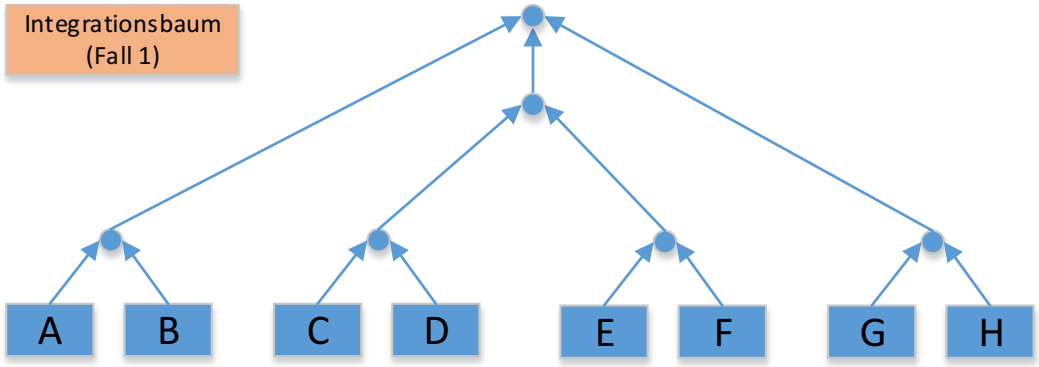






Integrations-strategie 4 (3 Level)

Integrationsbaum (Fall 1)



Integrationsbaum (Fall 2)

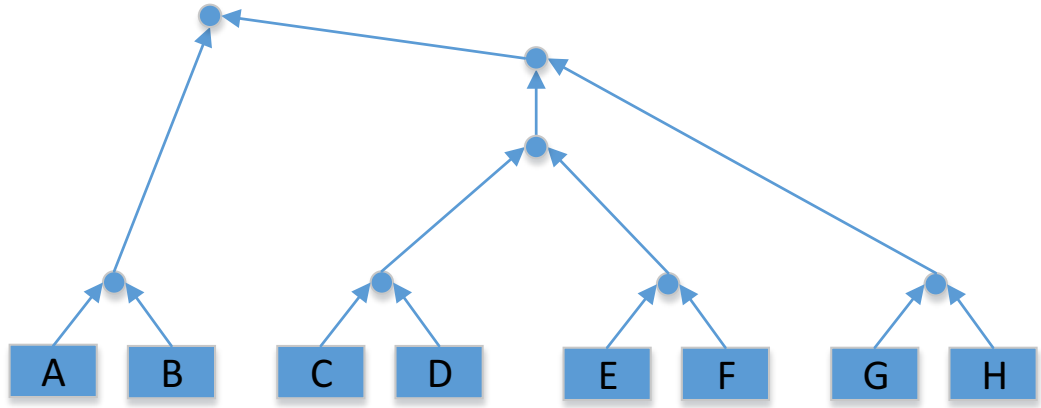


Abbildung zwischen Integrationsbaum (also der Strategie) und der Struktur sollte möglichst intuitiv sein.

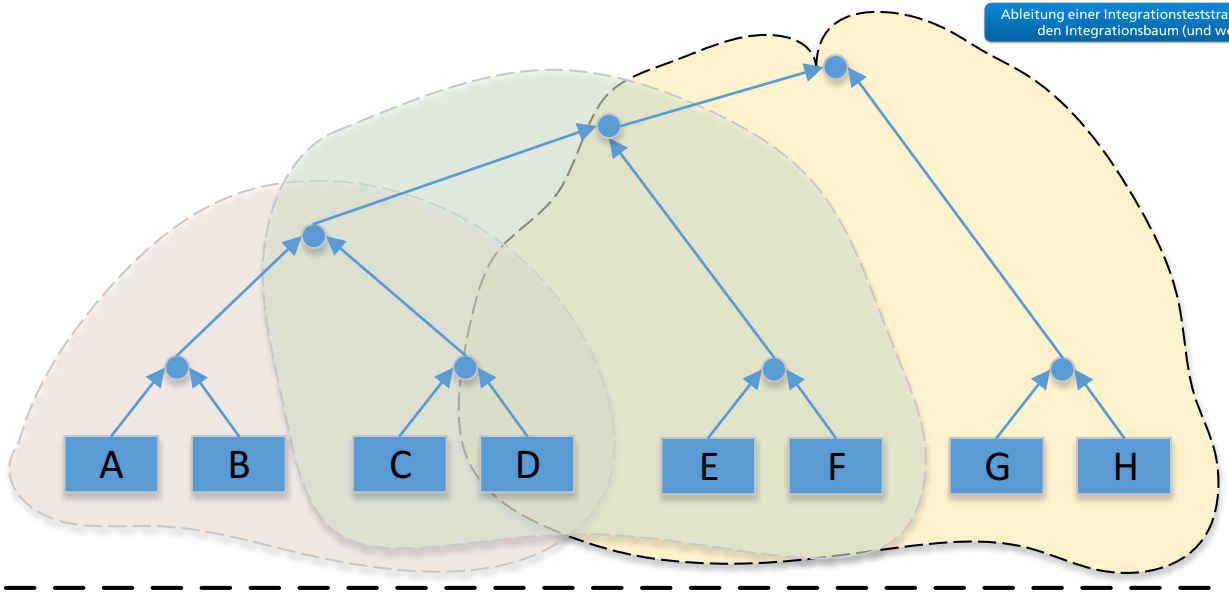
© Fraunhofer IESE

Vereinfachtes Beispiel



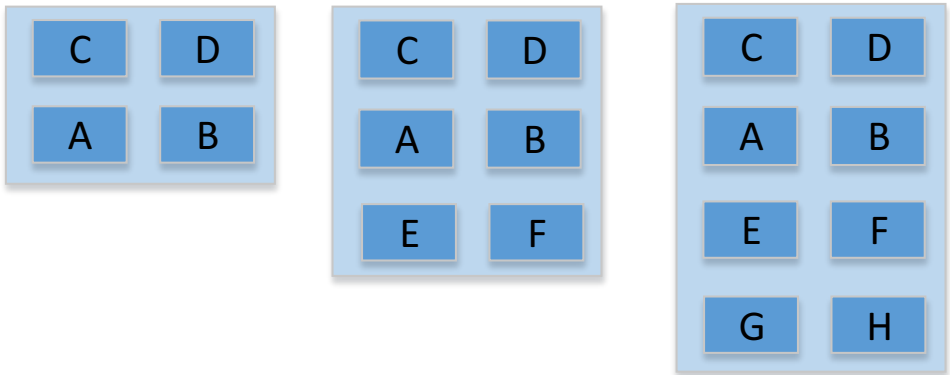
Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Integrationsbaum



Use Cases

Integrationsstrategie 5 (1 Level)



Use Case 1 | Use Case 2 | Use Case 3

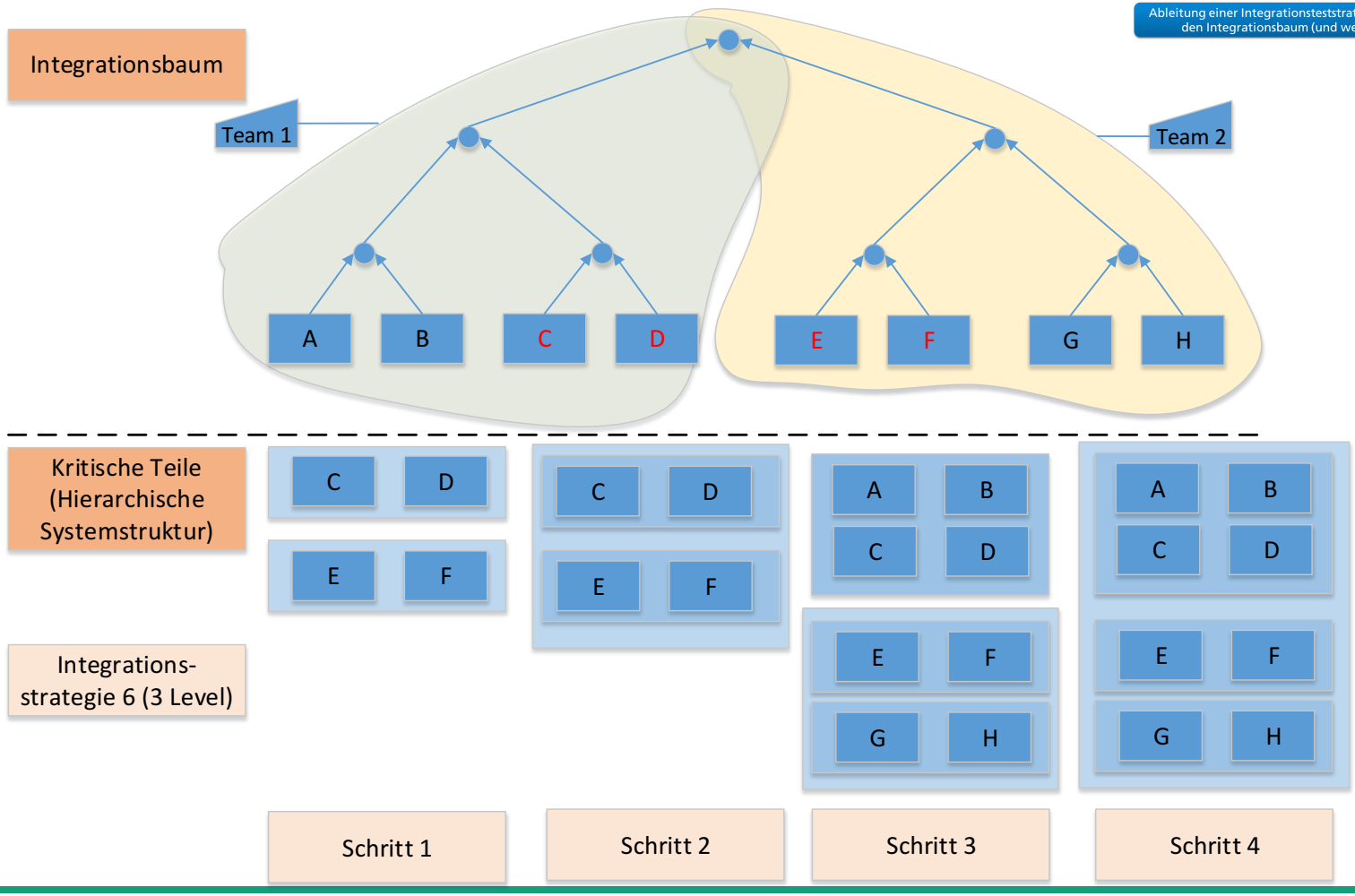
© Fraunhofer IESE

Fraunhofer IESE

Vereinfachtes Beispiel



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).



© Fraunhofer IESE

Fraunhofer IESE



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Der Integrationsbaum unterstützt bei der Ableitung einer Integrationsteststrategie, bietet aber keine 1:1 Abbildung.

© Fraunhofer IESE



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Neben den Strukturinformationen können u.a. Risiko, Datenflüsse, Topologie oder funktionale Dekomposition einen Einfluss auf die Strategie haben.

© Fraunhofer IESE



Ableitung einer Integrationsteststrategie unterstützt durch den Integrationsbaum (und weiterer Faktoren).

Pro Release muss unter Umständen der Integrationsbaum angepasst werden.

© Fraunhofer IESE

Der Integrationstest ist eine anerkannte QS-Phase während der Softwareentwicklung, die häufig jedoch nur untergeordnet betrachtet wird

Frühe Zusammenarbeit zwischen Tester und Architekt sollte forciert werden
(ergänzt um einen Projektleiter)

Explizit machen von Architekturelementen mittels eines Integrationsbaums als Grundlage für den Integrationstest

Unterstützung bei der Ableitung einer Integrationsteststrategie sowie Ergänzung weiterer Informationen für die Verfeinerung der Strategie (z.B. Risiko, Topologie, Funktionalität)

© Fraunhofer IESE



© Fraunhofer IESE

Spielt der Integrationstest tatsächlich eine untergeordnete Rolle – was sind Ihre Erfahrungen?

Wie führen Sie einen (effektiven / effizienten) Integrationstest durch?

Können hier vorgestellte Konzepte umgesetzt bzw. einen Mehrwert bringen?

Was sind Herausforderungen, die Ihnen zum Thema Integrationstest einfallen?



© Fraunhofer IESE

Backup

© Fraunhofer IESE