17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

QUALITY ASSURANCE: TESTING

CHRISTIAN KAESTNER
CARNEGIE MELLON UNIVERSITY
INSTITUTE FOR SOFTWARE RESEARCH

READING ASSIGNMENT NOV 16

Textbook, Chapter 10

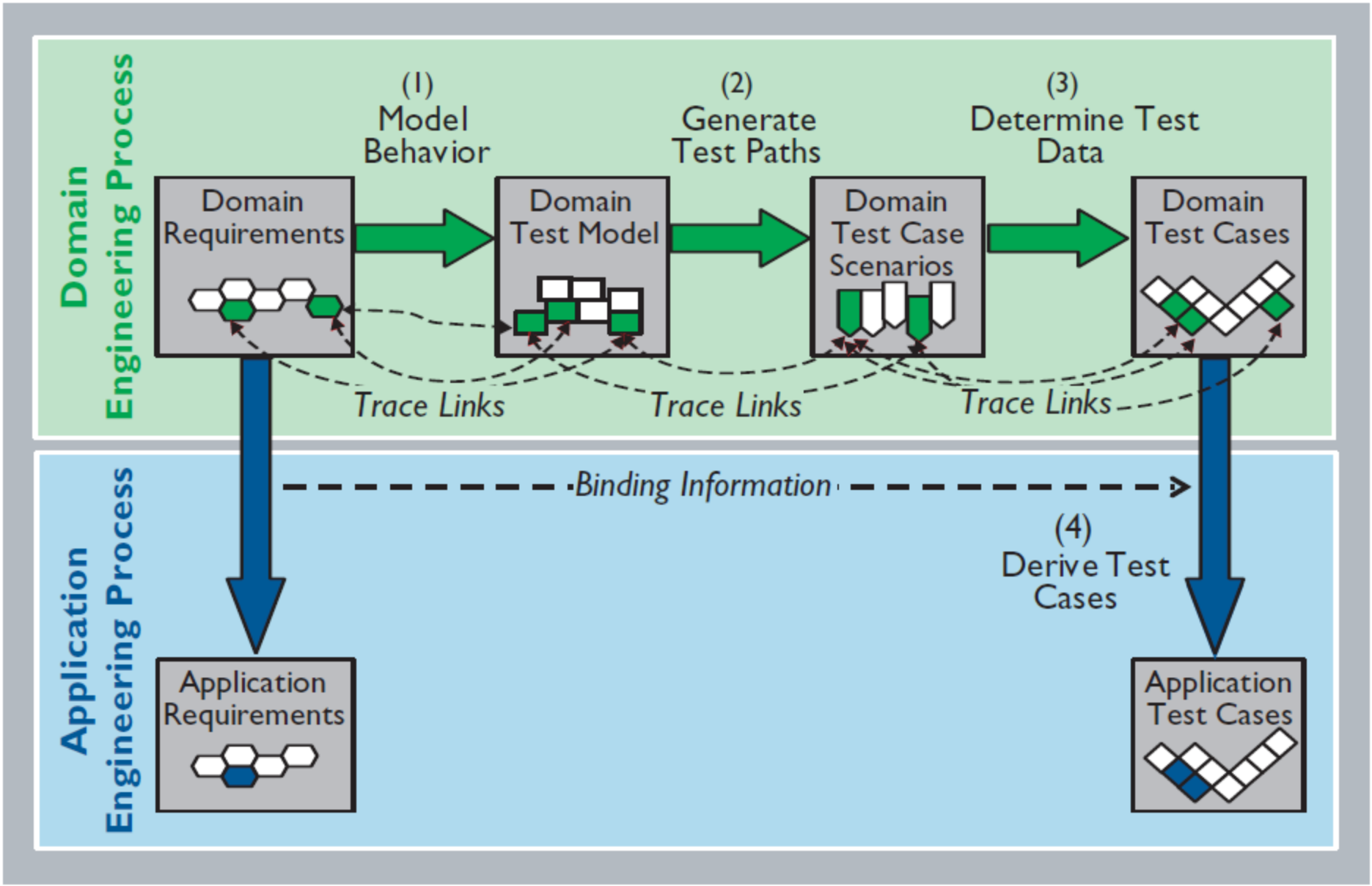
Thaker et al. Safe Composition. GPCE 2007

LEARNING GOALS

Understand what can be tested and how tests can be reused in a product line

Develop conditional tests in domain engineering and perform testing in domain engineering, understanding the limitations

Select a suitable test strategy for a given project



Requirements-based SPL testing

Source: Klaus Pohl and Andreas Metzger. 2006. Software product line testing. *Commun. ACM* 49, 12 (December 2006), 78-81

PRINCIPLES FOR SPL SYSTEM TESTING

P-1: Preserve Variability in Domain Test Artifacts

P-2: Test Commonalities in Domain Engineering

P-3: Use Reference Applications to Determine Defects in Frequently Used Variants

P-4: Test Commonalities based on a Reference Application

P-5: Test Correct Variability Bindings

P-6: Reuse Application Test Artifacts across Different Applications

Source: Klaus Pohl and Andreas Metzger. 2006. Software product line testing. *Commun. ACM* 49, 12 (December 2006), 78-81

DOMAIN TESTING

Domain Unit Testing

Domain Integration Testing

Domain System Test

Variability in Test Artifacts

Domain Test Artefact Reuse

Application Test Coverage

Application-specific Tests

Source: Pohl et al. Software Product Line Engineering. Chapters Domain Testing & Application Testing. Springer 2005

TEST STRATEGIES

Brute Force Strategy

Pure Application Strategy

Sample Application Strategy

Commonality and Reuse Strategy

- Domain testing aims at testing common parts and preparing test artefacts for variable parts. Application testing aims at reusing the test artefacts for common parts and reusing the predefined, variable domain test artefacts to test specific applications.

	Time to create	Absent variants	Early validation	Learning effort	Overhead
(BFS)	-	-	+	0	-
(PAS)	0	+	-	+	-
SAS	0	+	+	+	-
CRS	+	+	0	-	+
Combined SAS/CRS	+	+	+	0	0

Source: Pohl et al. Software Product Line Engineering. Chapter Domain Testing. Springer 2005

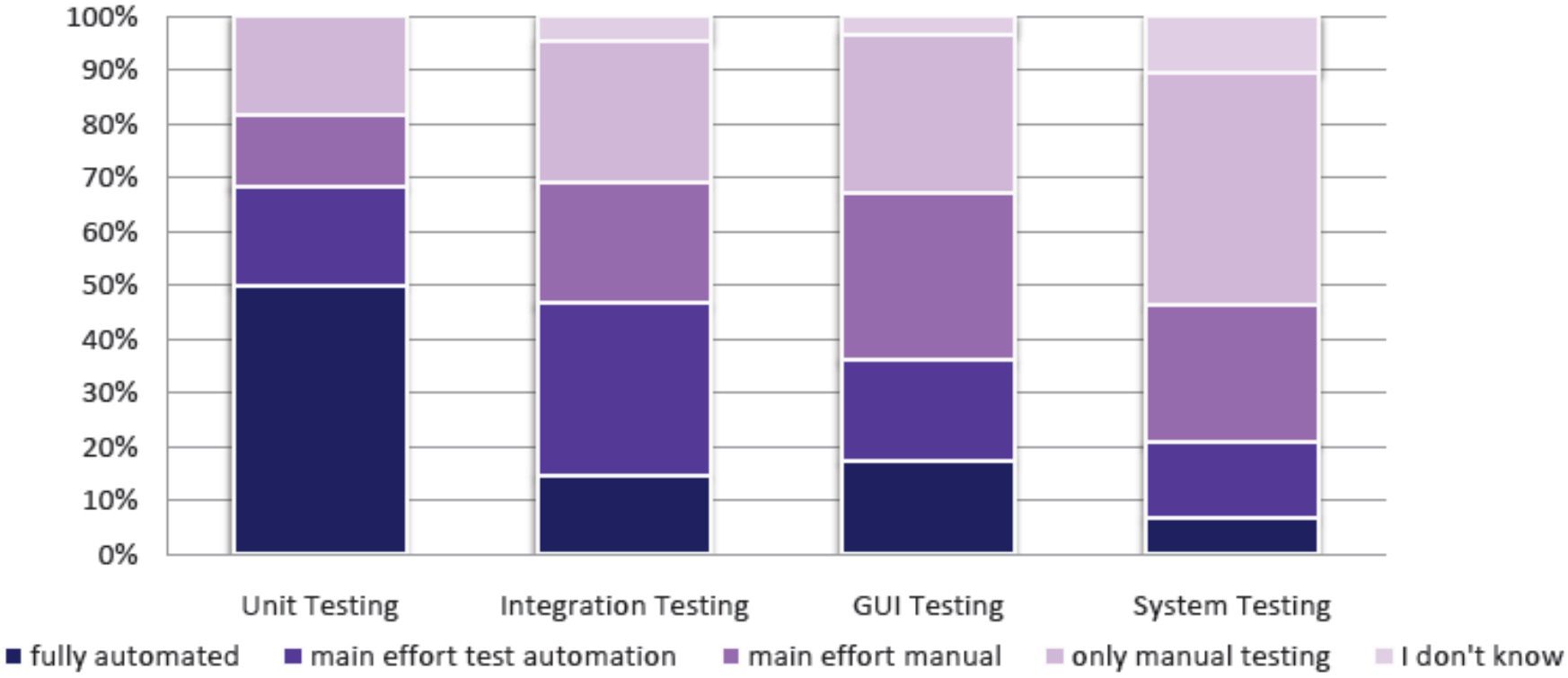


Figure 1. Test automation for each test practice

Source: Greiler, Michaela, Arie Van Deursen, and Margaret-Anne Storey. "Test confessions: a study of testing practices for plug-in systems." *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012.

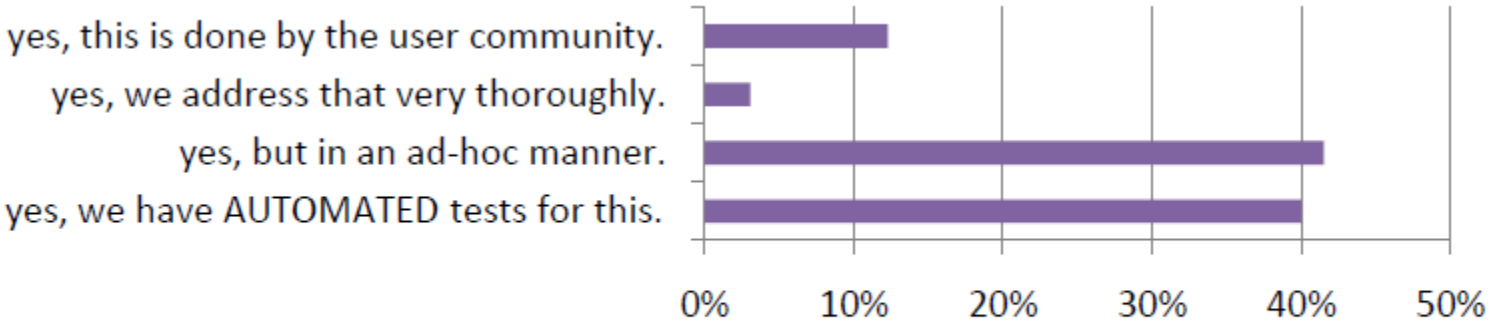


Figure 2. Cross-Product Integration Testing

Source: Greiler, Michaela, Arie Van Deursen, and Margaret-Anne Storey. "Test confessions: a study of testing practices for plug-in systems." *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012.

OPEN VS CLOSED WORLD

What's the Specification?

Typically **global** property x for every program

- Syntactically correct, well-typed
- Absence of double-free vulnerabilities
- Returns positive number for parameter 3
- Terminates within 10 seconds

Challenge is checking all configurations

- e.g., $\forall p \in PL: p \models x$

Brute-Force

Product Configuration

Conventional Analysis

Feature-Based Specifications

Property x for every program with feature f

No access to the file system

Renders "[:weather:]" as

Challenge is checking many configurations

e.g., $\forall p \in PL: (f \in p) \Rightarrow (p \vDash x)$

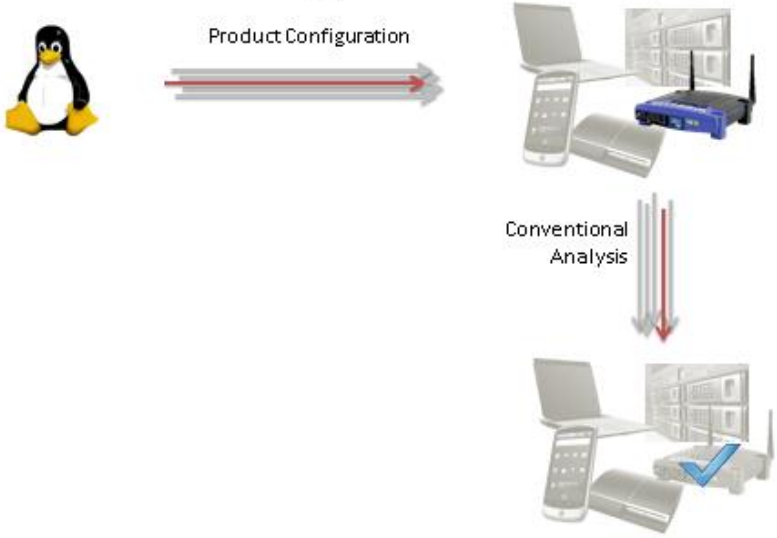
[Apel et al., COMNET'13]

AG (incoming(email *e*) & e.isEncrypted =>
((outgoing(email *e*) => e.isEncrypted) **R** outgoing(email *e*))

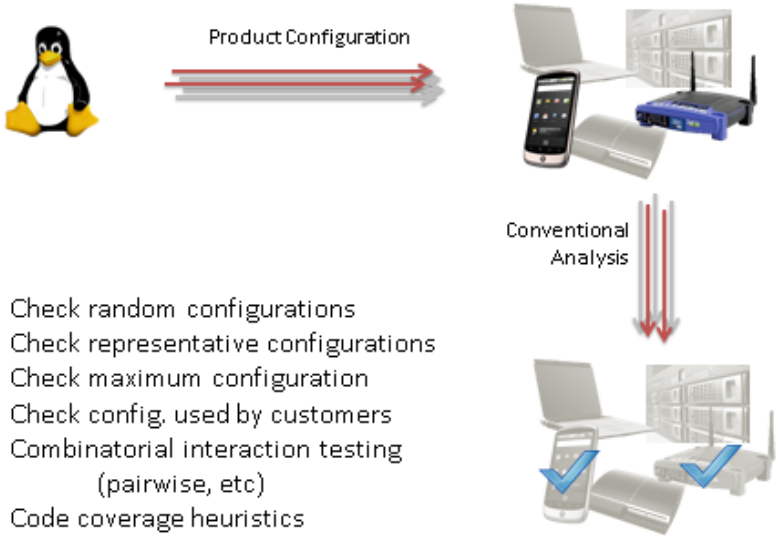
Feature-Based Specification

Brute-Force

One Configuration



Sampling



Check random configurations
Check representative configurations
Check maximum configuration
Check config. used by customers
Combinatorial interaction testing
(pairwise, etc)
Code coverage heuristics

Complete Analysis