TSP Symposium 2012
St. Petersburg, Florida, USA, September 17-20, 2012

Integrating Model-Driven Engineering Techniques in the Personal Software Process

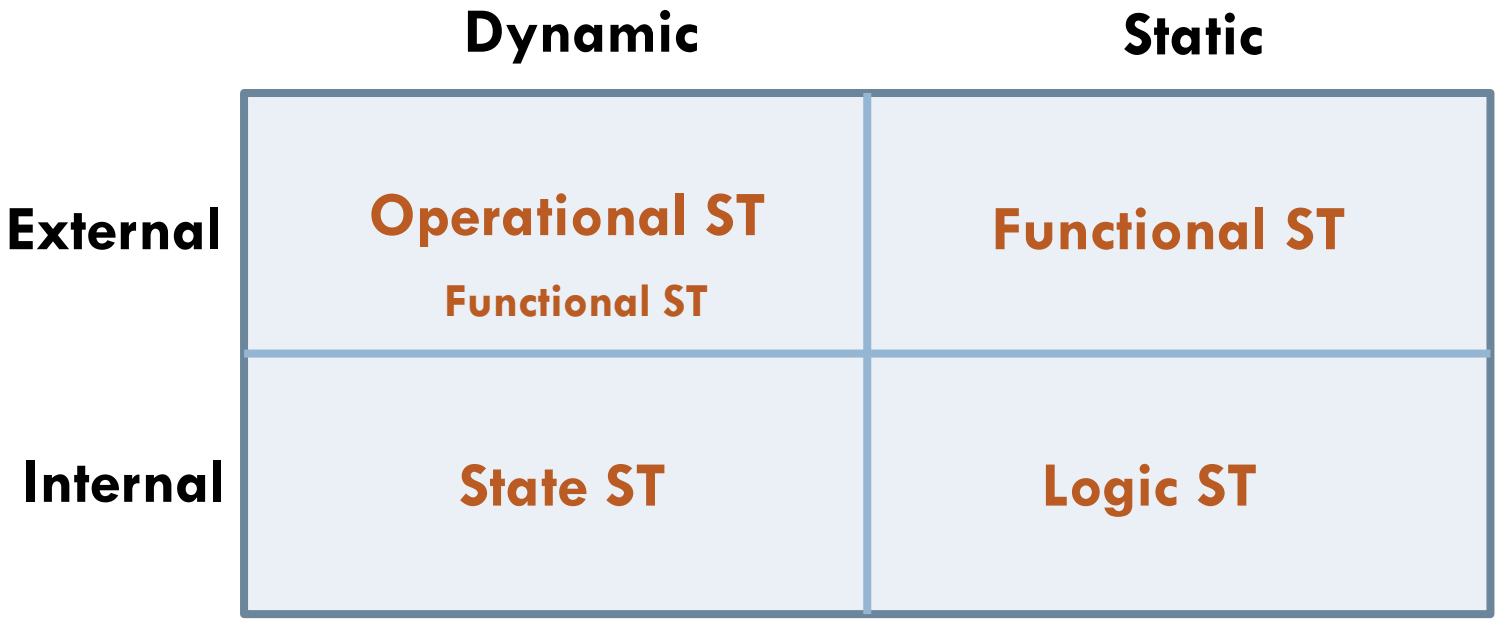
João Pascoal Faria
Faculty of Engineering, University of Porto, Portugal
(jpf@fe.up.pt)

Index

- Background and motivation: PSP, UML, MDE
- A lightweight MDE approach
- Key features and benefits
- Refinements to PSP scripts
- Lessons learned from case studies
- Conclusions and future work

Background and motivation: PSP design specification templates

- The PSP provides a set of **specification templates** for **completely** and **precisely** recording **reviewable** software designs covering 4 important design **views**



Background and motivation: PSP & UML

- UML is a standard visual notation for representing OO software designs, and is supported by many tools
- Especially when enriched with
  - contract specifications (pre/post conditions and invariants) in OCL
  - algorithm descriptions in a UML compliant action language
  - documentation notes and properties of relevant model elements

UML diagrams provide a convenient and familiar means for recording essentially the same info as the PSP templates

	Dynamic	**Static**
External	Uses cases and sequence diagrams	Class diagrams (+OCL or API doc)
Internal	Statemachine diagrams	Activity diagrams (flowcharts) or action specifications

Background and motivation: MDE (1)

- Although PSP is agnostic about the usage given to design specs/models: as documentation or compilable artifacts …
- …UML practitioners have concluded that building detailed design models for documentation only has several problems
  - is time consuming
  - the resulting models are often wrong
    - lack of static analysis, compilation, execution, etc., to spot problems
  - the resulting models soon become outdated and are not maintained
- Recent Model-Driven Engineering (MDE) approaches aim at avoiding such problems by generating code from models
  - If not production code (MDD), at least test code (MBT)

Background and motivation: MDE (2)

- In fact, with that MDE approach (code generation from models)
  - the time invested in building design models can be recovered
  - the quality of the models can be checked
  - there are higher chances that models are kept up to date
- This is also more in line with the agile values
  - (value more) *Working software over comprehensive documentation*
- This will also help solving problems we found when introducing PSP training in academia, using UML as the design notation:
  - Instructors time for grading and feedback is exacerbated when UML models are required for documentation only, because students don't have a reliable means to check by themselves if the models are right
  - Students see the cost of creating design models, but practically no short term benefits

A lightweight MDE approach (1)

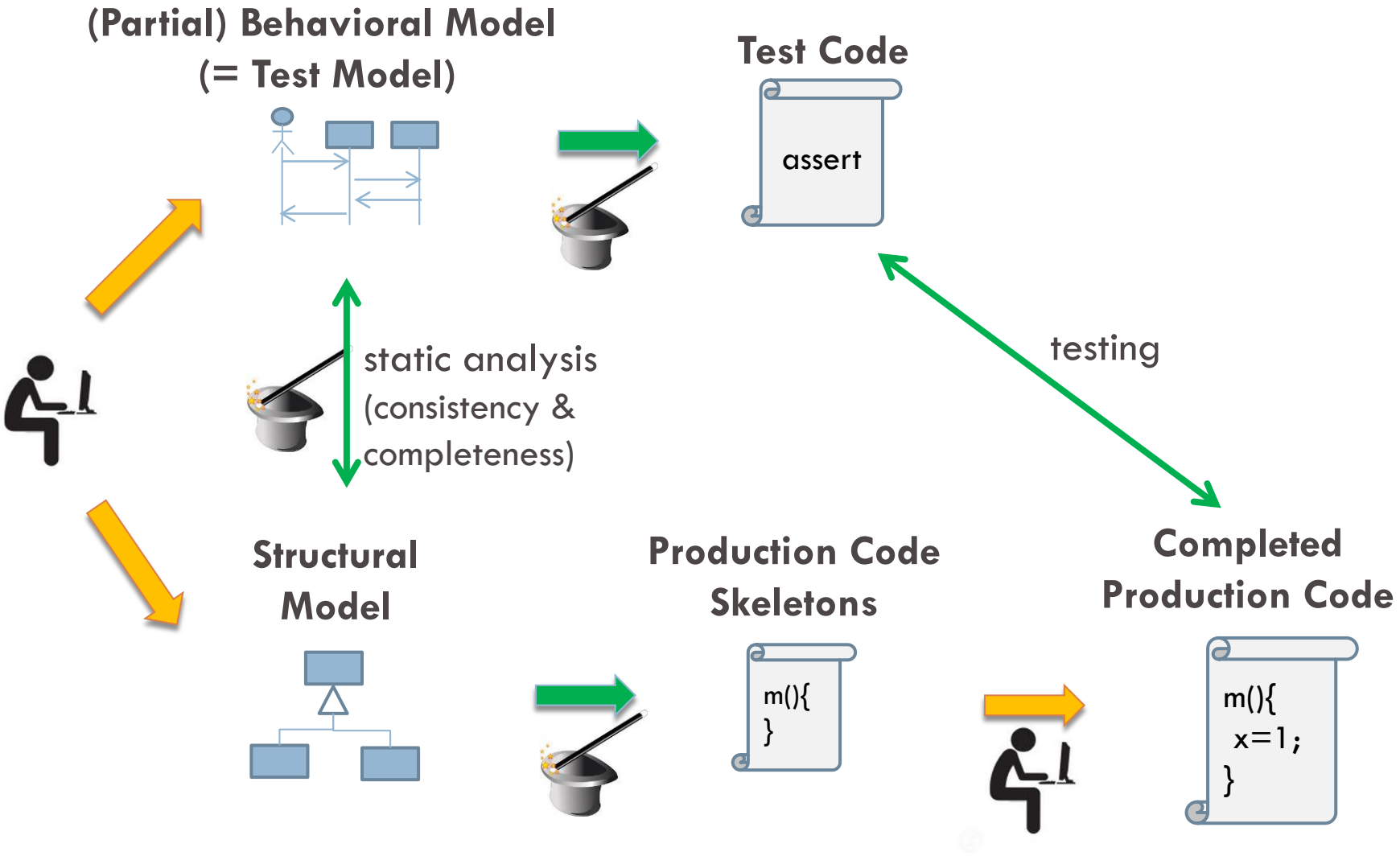
- Unfortunately, the level of detail of behavioral models needed to generate complete apps is often too high or only effective for specific domains (with domain specific languages)

- So, we propose a lightweight MDE approach:
  - develop structural models, from which parts of the application can be generated (e.g., class skeletons) (MDD)
  - develop *partial* behavioral models, not sufficient for app generation, but adequate for test generation (MBT)

- This is also inline with some agile practices (your tests are your specs, or vive-versa)

Partial behavior spec = Test spec

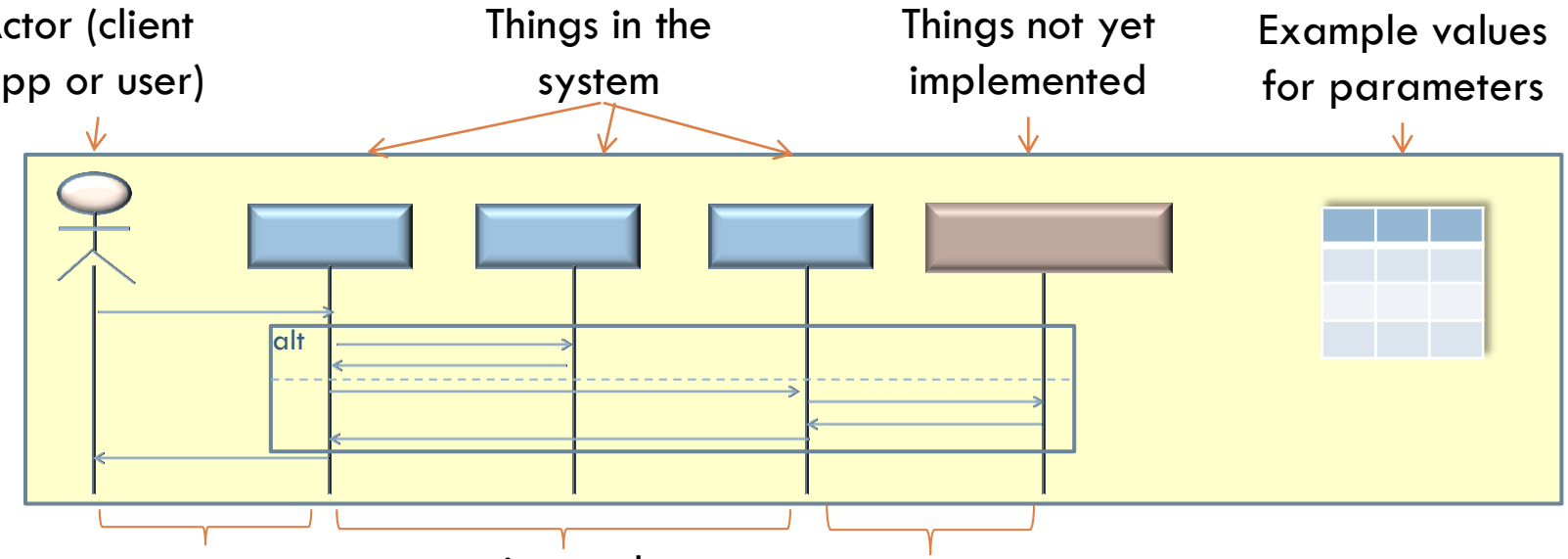
A lightweight MDE approach (2)



Behavior modeling and testing

(at all levels: unit, integration, system)

Behavioral Model/Spec



Generated Test Code

(Driver) Generate inputs as in spec and check responses against spec

(Monitor) Trace execution and check against spec

(Stub) Generate the responses as in spec

Exercise the scenario for each example

Process & tools

DLD
(incl. test spec.)

DLDR
(incl. static analysis)

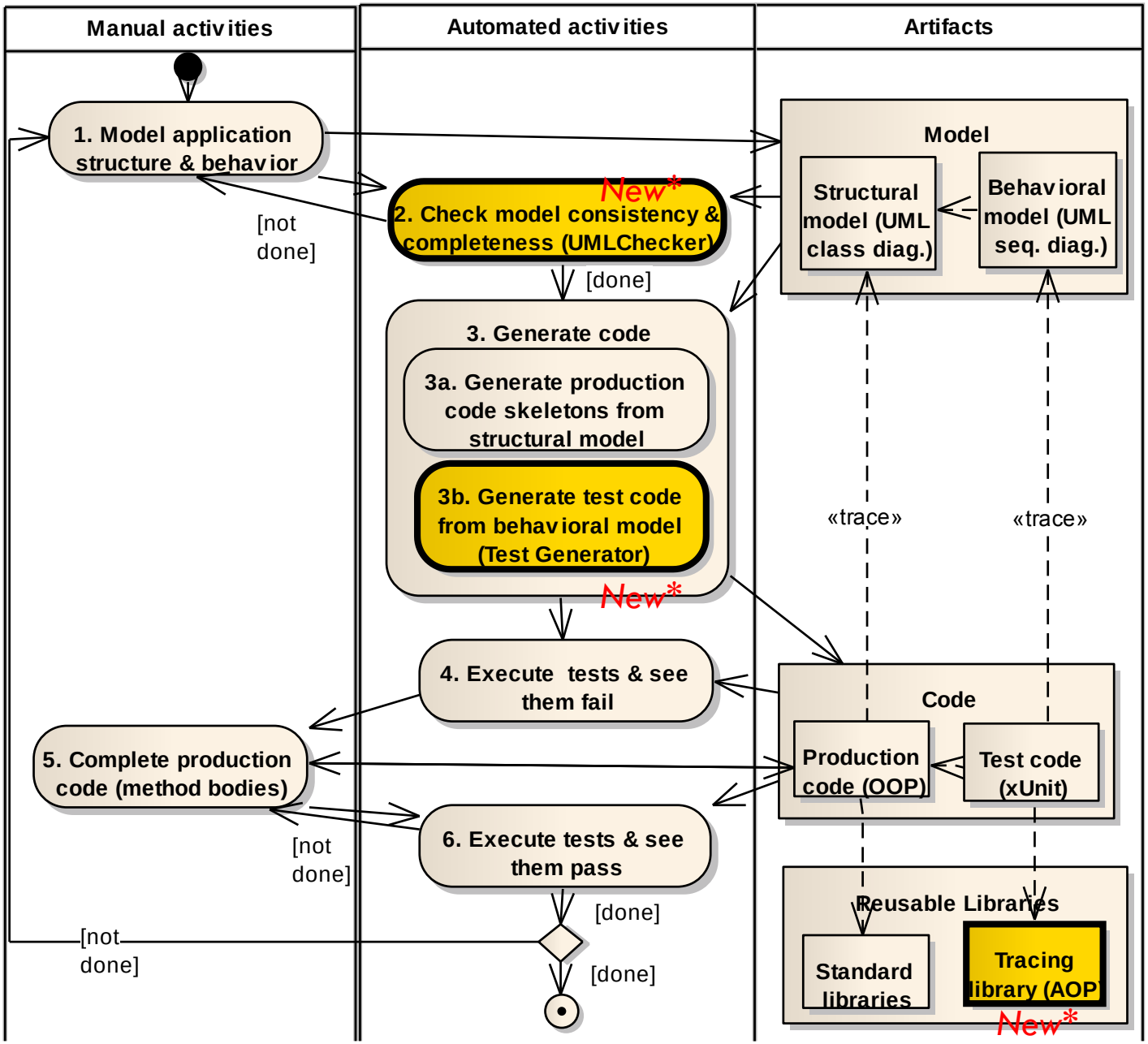
CODE

UT

CODE
CR

UT

* J. Faria, A. Paiva, Z. Yang, Test Generation from UML Sequence Diagrams, Proc. of the 8th Int. Conf. on the Quality of Information and Communication Technologies (QUATIC 2012), IEEE CPS, 2012



Key features and benefits (1)

Feature

- Support the modeling & automatic testing of
  - External interactions with users (UI)
  - External interactions with client applications (API)
  - Internal interactions among objects in the program

Benefits

- Covers the 4 design views (w/ structural model)
- Assures higher conformance with spec
- Improves fault localization
- Accelerates test phase

	Dynamic	Static
Ext.	Sequence diagrams (external interactions)	Class diagrams (public/external interfaces)
Int.	Sequence diagrams (internal interactions)	Class diagrams (private/internal interfaces)

Key features and benefits (2)

Feature	*Benefits*
▫ Parameterization ▫ Combined fragments (alt, opt, loop, par)	▫ Keep behavioral specs as generic as desired
▫ Loose conformance checking ▫ additional or intermediate calls are allowed in implementation	▫ Keep behavioral specs as simple as desired (focus on relevant interactions)
▫ Automatic checking of model consistency & completeness	▫ Verifiable completeness criteria ▫ Higher quality assurance
▫ "Stubs" inject the specified response messages for things marked as not yet implemented	▫ Iterative implemention & testing ▫ Independence of external components

Refinements to PSP scripts (1/2)

PSP2.1 Development Script

Purpose	To guide the development of small programs
Entry Criteria	– …

Step	Activities	Description
1	Design	– Review the requirements and produce an external specification to meet them. – ~~Complete Functional and Operational Specification templates to record this specification.~~ – **Develop a design model to describe externally visible system structure and behavior. (*)** – Produce a design to meet this specification. – ~~Record the design in Functional, Operational, State, and Logic Specification templates.~~ – **Refine the design model to describe internal system structure and behavior. (*)** – (…)

(*) Guidelines about diagrams, templates, completeness criteria, etc., in **Design standard**

Refinements to PSP scripts (2/2)

PSP2.1 Development Script (cont.)

Step	Activities	Description
2	Design Review	– Follow the Design Review script and checklist and review the design. – **Check the design model with a static analysis tool.** – ….
3	Code	– **Generate initial production code from the design model.** – …
4	Code Review	
5	Compile	
6	Test	– **Generate initial test code from the design model.** – …

Exit Criteria	– A thoroughly tested program that conforms to the Coding standard ~~– Completed Design templates~~ – **Completed design model consistent with the code** – …

Lessons learned from case studies

- We validated the approach viability on a set of case studies
- Size metrics and savings are promising, as in a typical example:

Item	Size unit	Manual	Generated
Structural model	model elements	42	0
Behavioral model		56	0
Subtotal		98	0
Production code	LOC	174	81
Test code		0	82
Subtotal		174	163

- We also found some manageable issues
  - Compilable models still need some doc. notes for human readbility
  - More details fixed in design than usual
  - Very small iterations are problematic (same as for metrics collection)

Conclusions

- Presented a lightweight MDE approach
  - Based on lightweight behavioral and structural models
  - (Partial) production code and (full) test code generation from models
- That is "PSP friendly"
  - Covers the 4 design views (in a sense of "internal")
  - Promotes complete (in a sense), precise and reviewable designs
  - Implies minimal changes to design scripts
  - Embeds test specification in the design phase (as behavior specs)
  - Is designed to bring short term productivity and quality benefits
- And "agile friendly"
  - Compilable models are not mere documentation
  - TDD/BDD [create a test = create an (external + internal) behavior spec]

Future work

- Conduct more extensive experiments, using the PSP measurement framework, to quantify the productivity & quality gains and better understand the contexts of applicability
- Devise a simplified way to specify exceptional behavior
- Extend the approach and tools to broaden its applicability
  - other target languages (now only Java)
  - other modeling tools (now only Enterprise Architect)
  - GUI testing (now, only command line interface testing), particularly for system testing
  - testing of time constrained, concurrent and distributed systems, particularly for integration testing

Thank You!

Questions?

Suggestions?