Testing of Reactive Systems

Lecture 1: Introduction

Henrik Bohnenkamp

Lehrstuhl Informatik 2 (MOVES)
RWTH Aachen

Summer Semester 2009

Testing of Reactive Systems

Overview

- Course in Theoretical Computer Science
- Prerequisites: Automata Theory (ATFS, FSAP)
- Language: English
- Complementary to Model-Checking (overlap in basics)

Presentation

- Some slides but very much on blackboard
- Concise script on the web page (.pdf)
- Plus accompanying slides and handouts

Testing of Reactive Systems

Overview

- Course in Theoretical Computer Science
- Prerequisites: Automata Theory (ATFS, FSAP)
- Language: English
- Complementary to Model-Checking (overlap in basics)

Presentation

- Some slides but very much on blackboard
- Concise script on the web page (.pdf)
- Plus accompanying slides and handouts

Organisation

- Lecture:
  - Mondays, 12:30 – 14:00, AH 3
  - Tuesdays, 11:45 – 13:15, AH 3
  - $\approx$ 18 sessions
- Exercise class:
  - Bi-weekly
  - Usually on Mondays instead of the lecture
  - Exception: first exercise Tuesday, 28. April
  - Sheets on the web page, one week in advance
- Theory course in “Hauptstudium Informatik Diplom”, “Master Informatik”, “Master Software Systems Engineering”
- Homework in Groups of at most 3 (for those who want to take the exam mandatory!)
- Written exam at end of semester (or oral exam, depending on demand)

Contact

My Coordinates

Henrik Bohnenkamp
E1, Room 4210
Tel 0241 80-21203
e-mail henrik@cs.rwth-aachen.de

In case of questions

- Pop in when you like
- Write an e-mail

Up-to-date information

http://www-i2.informatik.rwth-aachen.de/i2/testing09/

Correctness of Software Systems

Infamous Examples

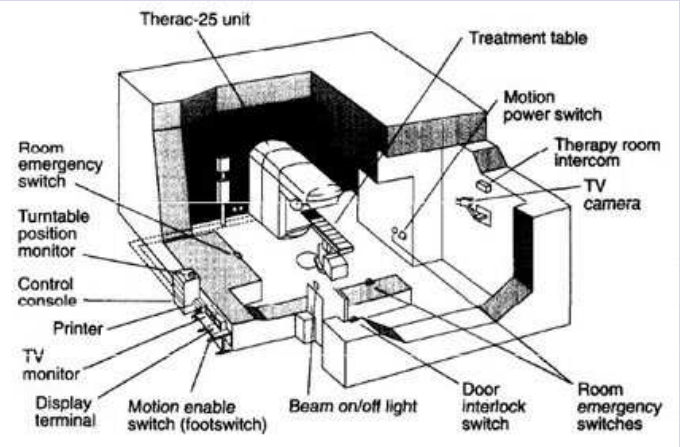
Ariane 5 disaster, 1996

- June 4, 1996
- **in essence: An integer overflow**

Correctness of Software Systems

Infamous Examples

Therac 25 accidents



- medical device for radiation treatment of cancer patients, 11 installed mid 1980's
- **radiation overdose**
- six patients injured or killed
- **in essence: a race condition overlooked**

Correctness of Software Systems

Heathrow Terminal 5: random quotes from the web

400,000 hours of software engineering went into the baggage handling system

Correctness of Software Systems

Heathrow Terminal 5: random quotes from the web

"180 IT suppliers and run 163 IT systems, 546 interfaces, more than 9,000 connected devices, 2,100 PCs and 'enough cable to lay to Istanbul and back' ".

Correctness of Software Systems

Heathrow Terminal 5: random quotes from the web

"Apparently the computer software told the baggage people that the flight had taken off. So everyone in the plane just watched as all their suitcases were taken back into the terminal instead of being loaded on."

Correctness of Software Systems

Heathrow Terminal 5: random quotes from the web

"They have been doing tests on the belt system for the last few weeks and knew it wasn't going right. The computer cannot cope with the number of bags going through."

Correctness of Software Systems

Heathrow Terminal 5: random quotes from the web

The system was tested for one year!

What are “Formal Methods”?

It is all about software and systems development

Primary Idea

- Writing a precise specification of a system
- Using a formal or mathematical syntax to do so
- Textual or graphical
- Semantics, i.e., a precise meaning of the language
- Formal specification eliminates ambiguity
- Reduces chance of errors during software development

What are “Formal Methods”?

It is all about software and systems development

Primary Idea

- Writing a precise specification of a system
- Using a formal or mathematical syntax to do so
- Textual or graphical
- Semantics, i.e., a precise meaning of the language
- Formal specification eliminates ambiguity
- Reduces chance of errors during software development

What are “Formal Methods”?

It is all about software and systems development

Primary Idea

- Writing a precise specification of a system
- Using a formal or mathematical syntax to do so
- Textual or graphical
- Semantics, i.e., a precise meaning of the language
- Formal specification eliminates ambiguity
- Reduces chance of errors during software development

What are “Formal Methods”?

It is all about software and systems development

Primary Idea

- Writing a precise specification of a system
- Using a formal or mathematical syntax to do so
- Textual or graphical
- Semantics, i.e., a precise meaning of the language
- Formal specification eliminates ambiguity
- Reduces chance of errors during software development

What are “Formal Methods”?

It is all about software and systems development

Primary Idea

- Writing a precise specification of a system
- Using a formal or mathematical syntax to do so
- Textual or graphical
- Semantics, i.e., a precise meaning of the language
- Formal specification eliminates ambiguity
- Reduces chance of errors during software development

What are “Formal Methods”?

It is all about software and systems development

Primary Idea

- Writing a precise specification of a system
- Using a formal or mathematical syntax to do so
- Textual or graphical
- Semantics, i.e., a precise meaning of the language
- Formal specification eliminates ambiguity
- Reduces chance of errors during software development

What are “Formal Methods”?

Purpose of Formal Methods

- Traditionally for formally verifying the correctness of software
- (Provably correct) Step-wise refinement from spec to code
- Code generation
- Model-checking

What are “Formal Methods”?

Purpose of Formal Methods

- Traditionally for formally verifying the correctness of software
- (Provably correct) Step-wise refinement from spec to code
- Code generation
- Model-checking

What are “Formal Methods”?

Purpose of Formal Methods

- Traditionally for formally verifying the correctness of software
- (Provably correct) Step-wise refinement from spec to code
- Code generation
- Model-checking

What are “Formal Methods”?

Purpose of Formal Methods

- Traditionally for formally verifying the correctness of software
- (Provably correct) Step-wise refinement from spec to code
- Code generation
- Model-checking

Benefits from Formal Methods

The process of creating a precise specification

- Allows articulation of a proper understanding of the system
- Reveals errors or aspects of incompleteness

Specification

- Can be analysed
- Can be verified correct against properties of interest (model-checking)
- . . . "Potentially automatic analysis of the relationship between the specification and the source code."

Benefits from Formal Methods

The process of creating a precise specification

- Allows articulation of a proper understanding of the system
- Reveals errors or aspects of incompleteness

Specification

- Can be analysed
- Can be verified correct against properties of interest (model-checking)
- . . . "Potentially automatic analysis of the relationship between the specification and the source code."

Benefits from Formal Methods

The process of creating a precise specification

- Allows articulation of a proper understanding of the system
- Reveals errors or aspects of incompleteness

Specification

- Can be analysed
- Can be verified correct against properties of interest (model-checking)
- . . . "Potentially automatic analysis of the relationship between the specification and the source code."

Benefits from Formal Methods

The process of creating a precise specification

- Allows articulation of a proper understanding of the system
- Reveals errors or aspects of incompleteness

Specification

- Can be analysed
- Can be verified correct against properties of interest (model-checking)
- . . . "Potentially automatic analysis of the relationship between the specification and the source code."

Benefits from Formal Methods

The process of creating a precise specification

- Allows articulation of a proper understanding of the system
- Reveals errors or aspects of incompleteness

Specification

- Can be analysed
- Can be verified correct against properties of interest (model-checking)
- . . . "Potentially automatic analysis of the relationship between the specification and the source code."

Using Formal Methods

Essential

- Means and algorithms to reason about specification
- Tools-support required
- Whatever you do, it must be computable. . .
- . . . efficiently

Using Formal Methods

Essential

- Means and algorithms to reason about specification
- Tools-support required
- Whatever you do, it must be computable. . .
- . . . efficiently

What is testing?

Software testing is mostly about empirically checking correctness, by experimenting with the system-under-test.

Performing experiments on software systems in order to increase software quality

What is testing?

Software testing is mostly about empirically checking correctness, by experimenting with the system-under-test.

Performing experiments on software systems in order to increase software quality

Goal of Testing?

Many attempts to answer this question:

1. . . . to show that the software is bug free
2. . . . to show that the software does what it is supposed to do
3. . . . to establish confidence that the system does as intended
4. . . . to find errors

Goal of Testing?

Many attempts to answer this question:

1. . . . to show that the software is bug free
2. . . . to show that the software does what it is supposed to do
3. . . . to establish confidence that the system does as intended
4. . . . to find errors

1

No. Testing is inherently incomplete.

Goal of Testing?

Many attempts to answer this question:

1. . . . to show that the software is bug free
2. . . . to show that the software does what it is supposed to do
3. . . . to establish confidence that the system does as intended
4. . . . to find errors

1

Testing can show the presence of bugs, but not their absence [Dijkstra]

Goal of Testing?

Many attempts to answer this question:

1. . . . to show that the software is bug free
2. . . . to show that the software does what it is supposed to do
3. . . . to establish confidence that the system does as intended
4. . . . to find errors

2

It can do this sometimes, but that is not enough.

Goal of Testing?

Many attempts to answer this question:

1. . . . to show that the software is bug free
2. . . . to show that the software does what it is supposed to do
3. . . . to establish confidence that the system does as intended
4. . . . to find errors

3

It can never establish confidence, only increase

Goal of Testing?

Many attempts to answer this question:

1. . . . to show that the software is bug free
2. . . . to show that the software does what it is supposed to do
3. . . . to establish confidence that the system does as intended
4. . . . to find errors

4

Testing is the process of executing a program with the intent of finding errors.

[Meyers 1979]

So what is Testing?

Tretmans: Testing is

- a technical process
- performed by experimenting with or executing the software
- in a controlled environment
- following a specified procedure
- with the intent to measure the quality of the software
- demonstrating deviation of intended behaviour

Test Objects

Procedures, functions, classes

- terminating
- monolithic
- deterministic
- simple input/output behaviour
- simple interfaces
- complication: side-effects

Example

- Java methods
- Java classes
- Haskell program
- ...

Test Objects

Procedures, functions, classes

- terminating
- monolithic
- deterministic
- simple input/output behaviour
- simple interfaces
- complication: side-effects

Example

- Java methods
- Java classes
- Haskell program
- ...

Test Objects

Procedures, functions, classes

- terminating
- monolithic
- deterministic
- simple input/output behaviour
- simple interfaces
- complication: side-effects

Example

- Java methods
- Java classes
- Haskell program
- ...

Test Objects

Processes, Reactive Systems

- non-terminating
- composed of interacting components
- non-deterministic
- complex input/output behaviour
- different interfaces

Reactive Systems

GUIs, web browsers, . . .

Browser:

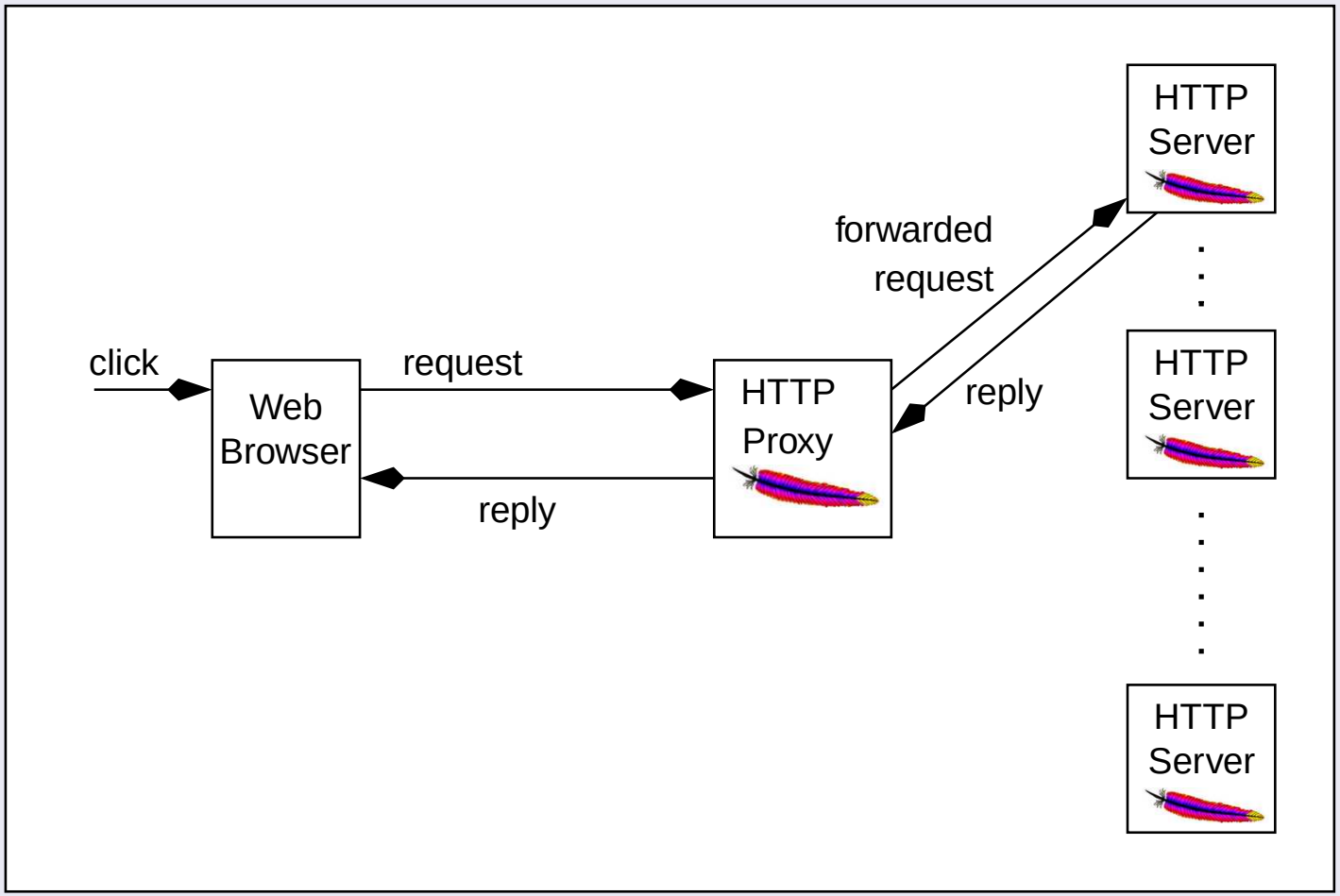
- Inputs: mouse-clicks
- Outputs: change of appearance, displayed information

But also:

- Outputs: HTTP-PDUs to web-servers or proxies
- Inputs: HTTP-PDUs from web-servers or proxies

Reactive Systems

web server



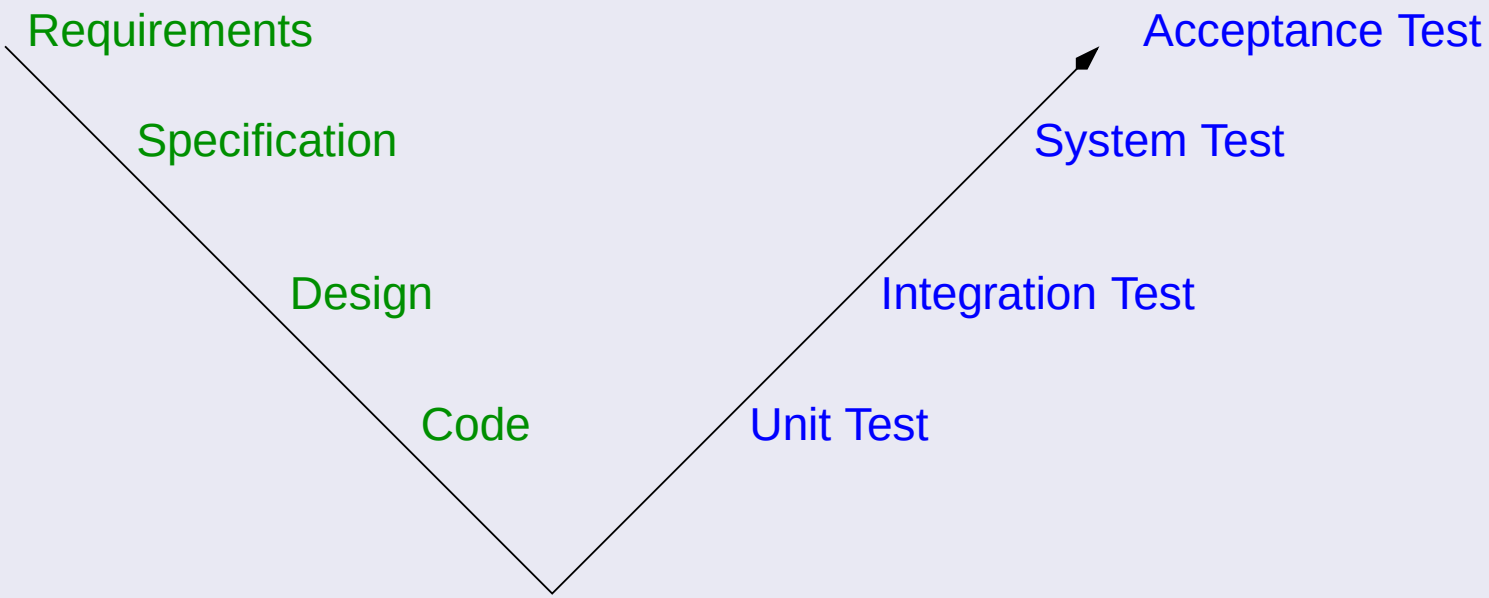
Reactive Systems

Vending Machine

- Inputs: Button presses, money
- Outputs: Coffee, Tea, Coke, Beer, Snacks

Testing, as traditionally done

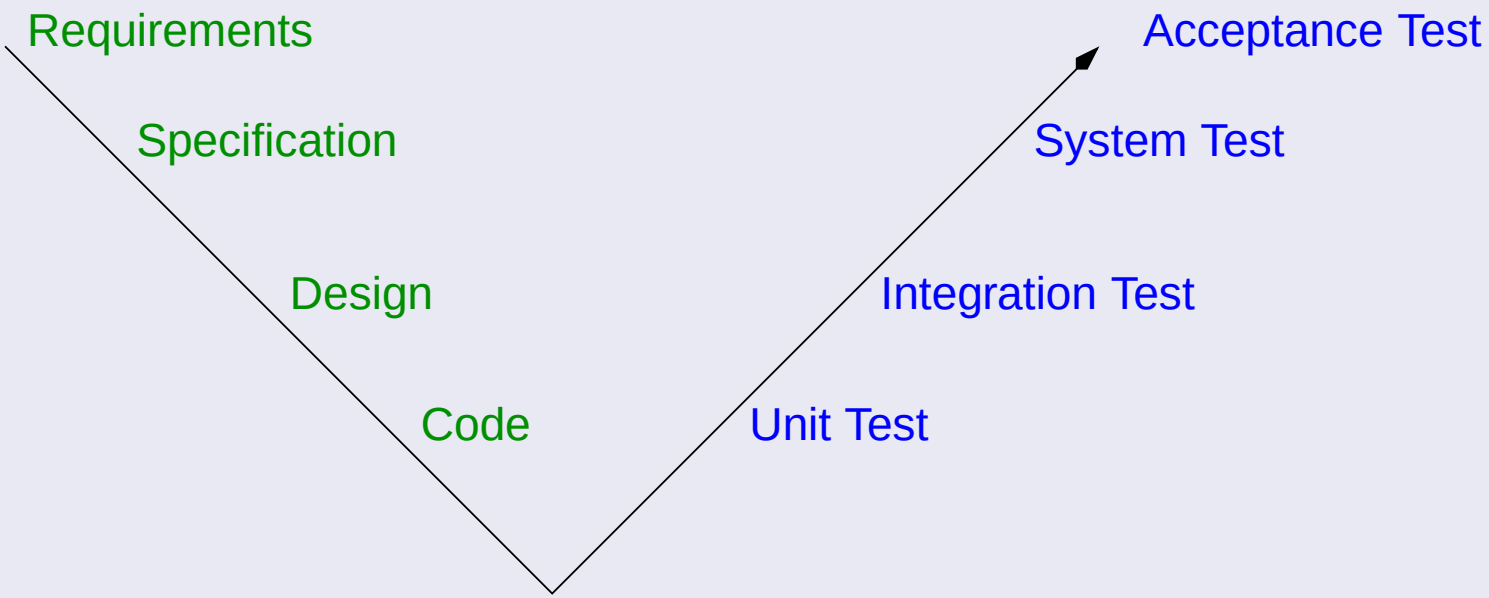
The (much simplified) V-Model



Requirements: obtained by analysing the needs of the user of the system to be developed

Testing, as traditionally done

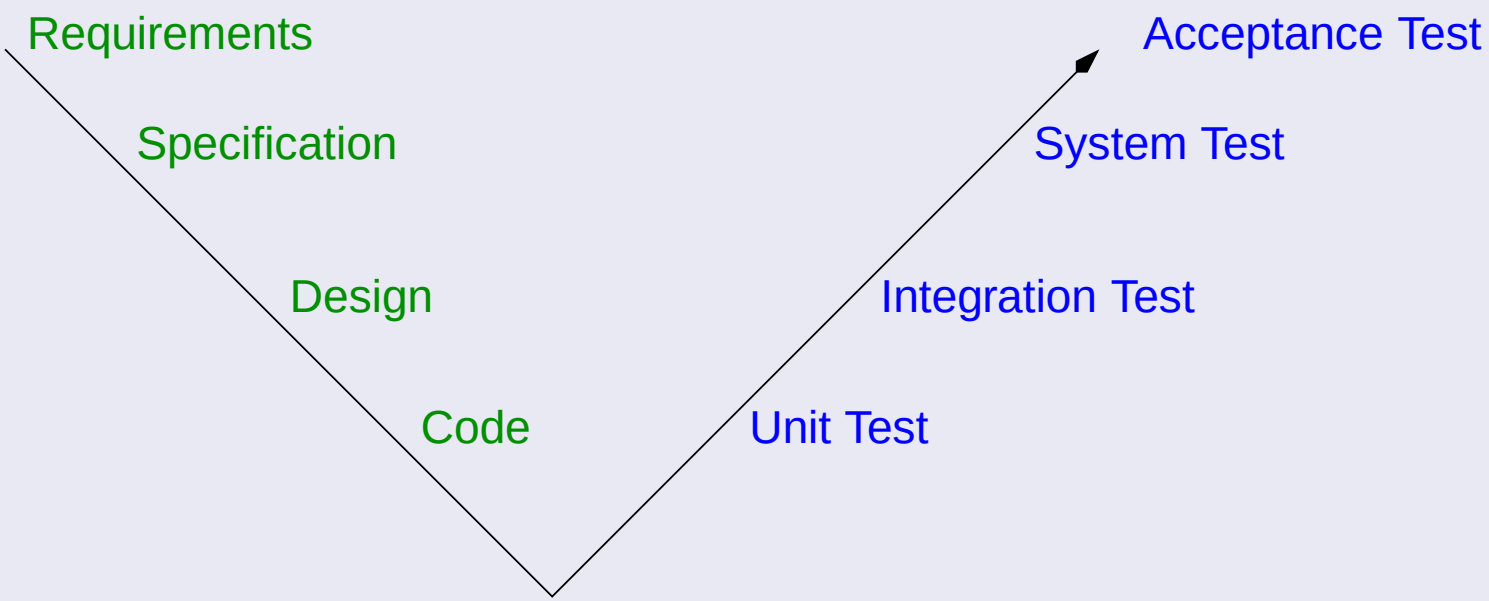
The (much simplified) V-Model



Specification: the functional and non-functional (performance, timing, etc.) properties the system must have in order to fulfill the requirements

Testing, as traditionally done

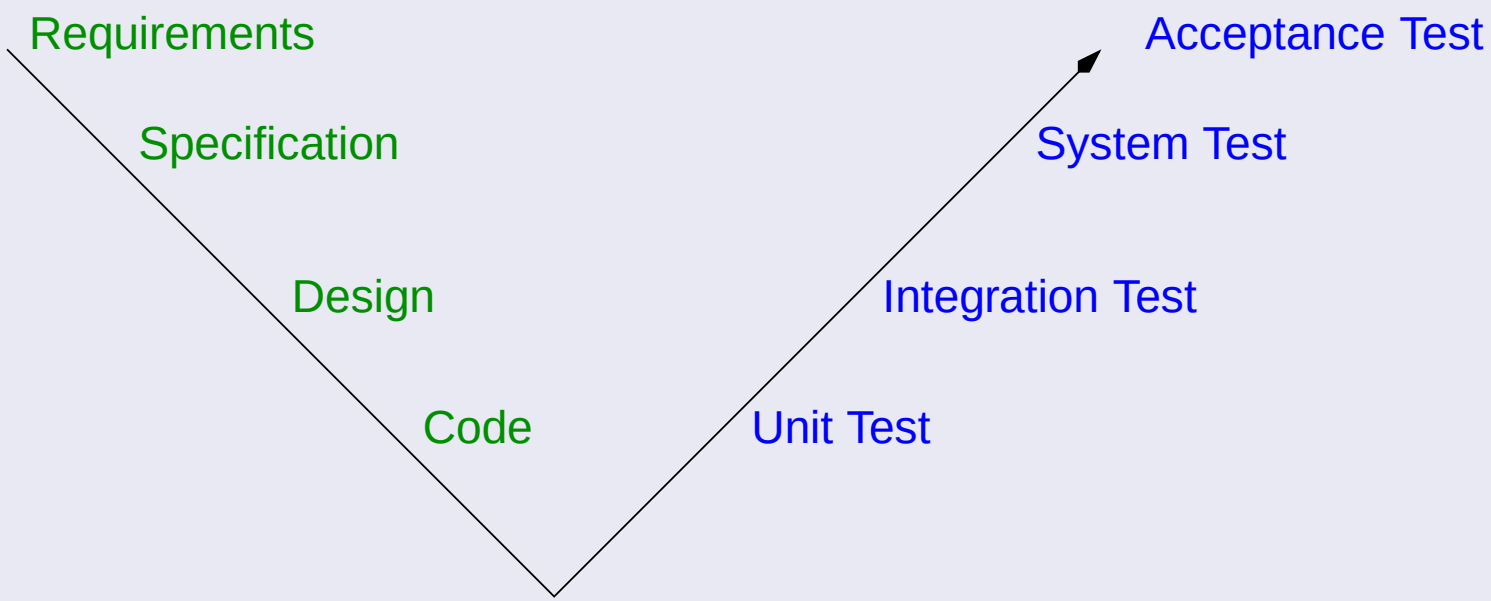
The (much simplified) V-Model



Design: description on how to structure the system, how components should interact via what interfaces etc.

Testing, as traditionally done

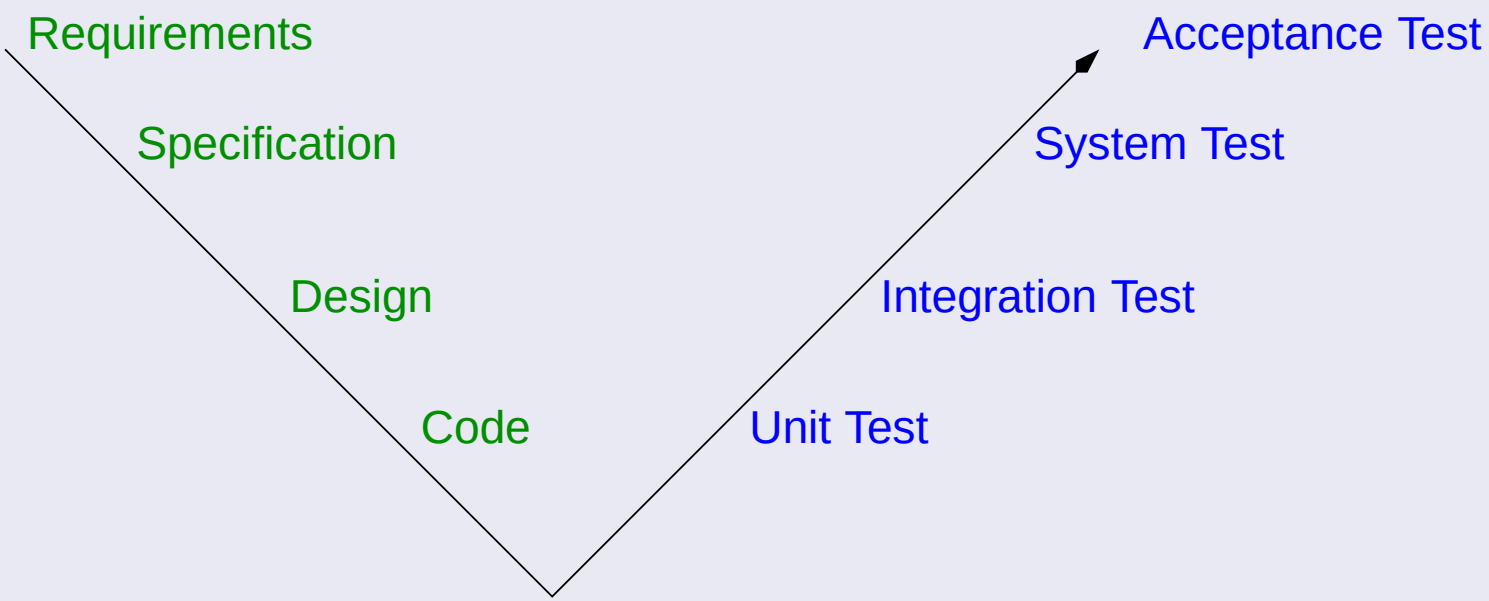
The (much simplified) V-Model



Code: the actual implementation of the system components

Testing, as traditionally done

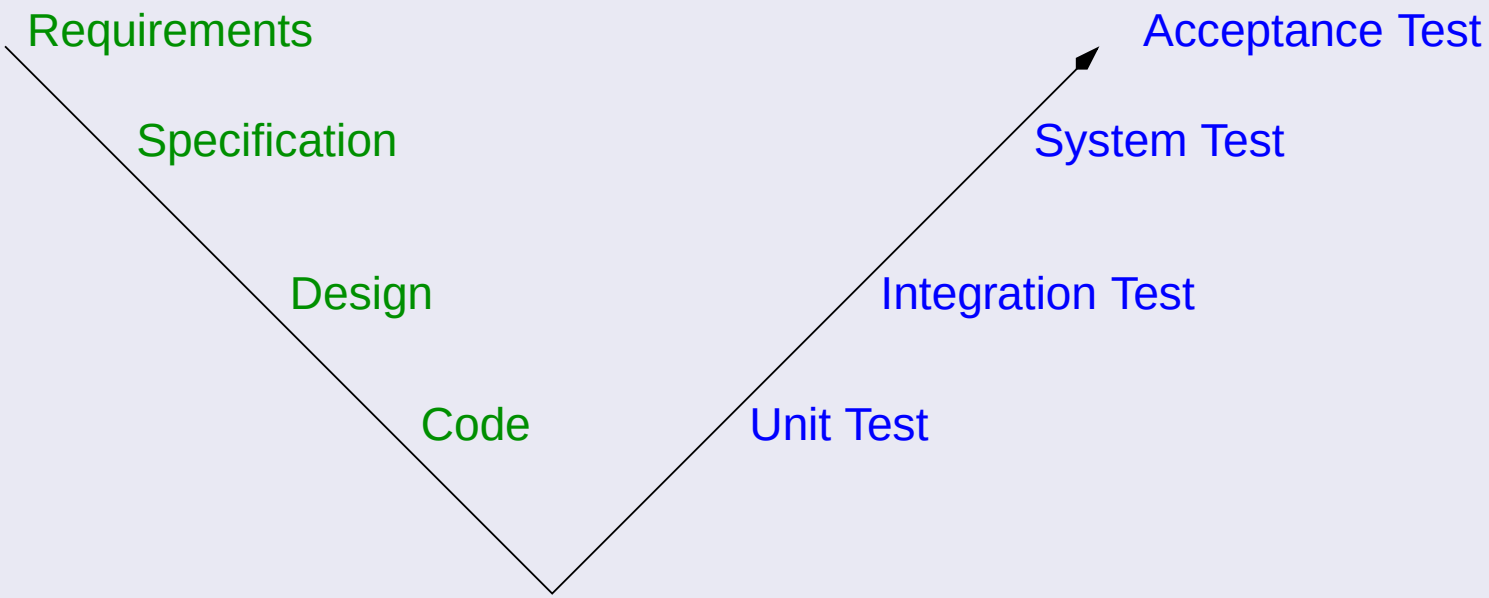
The (much simplified) V-Model



Unit Test: testing of individual components (methods, classes, functions, procedures,...)

Testing, as traditionally done

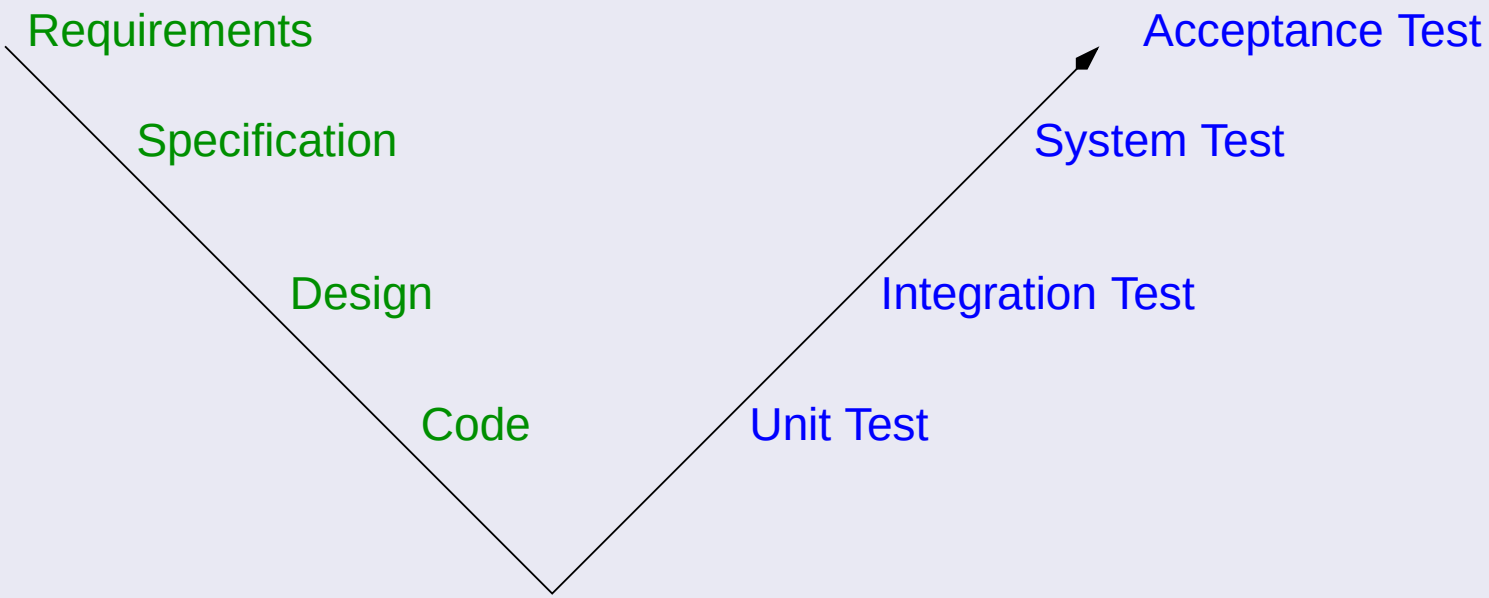
The (much simplified) V-Model



Integration Test: testing of larger aggregations of several interacting components

Testing, as traditionally done

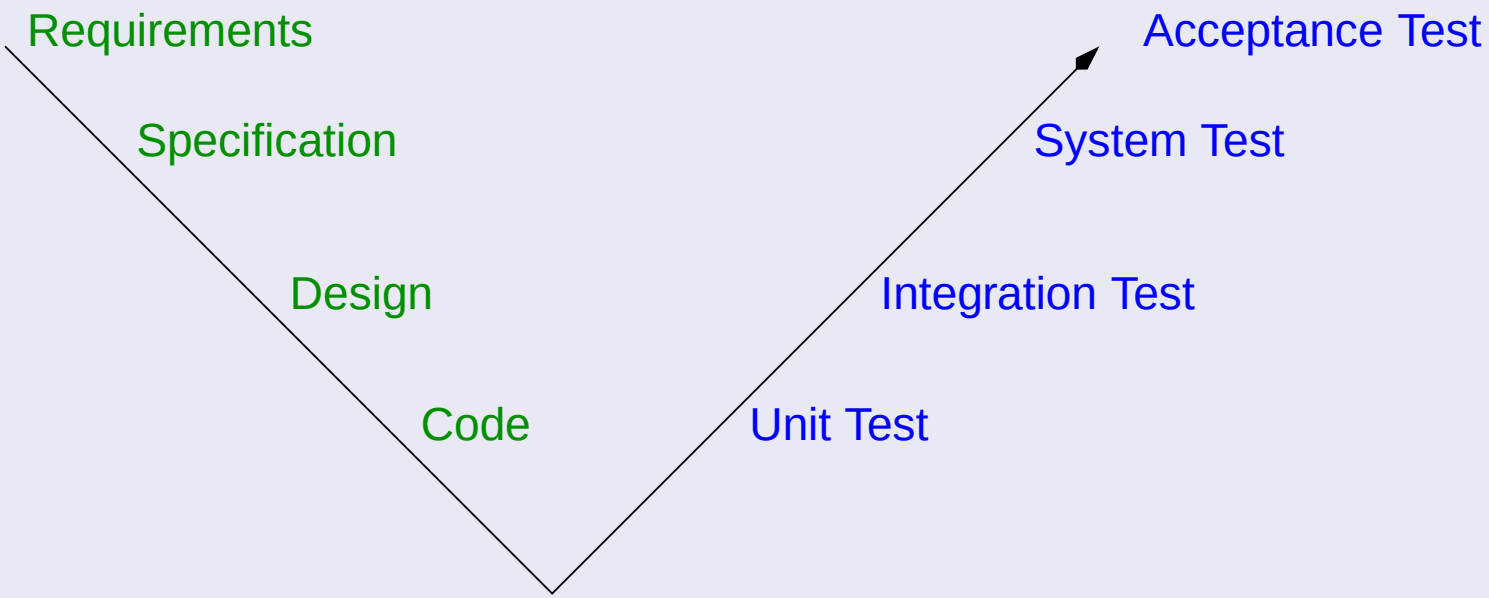
The (much simplified) V-Model



System Test: testing of completely integrated system

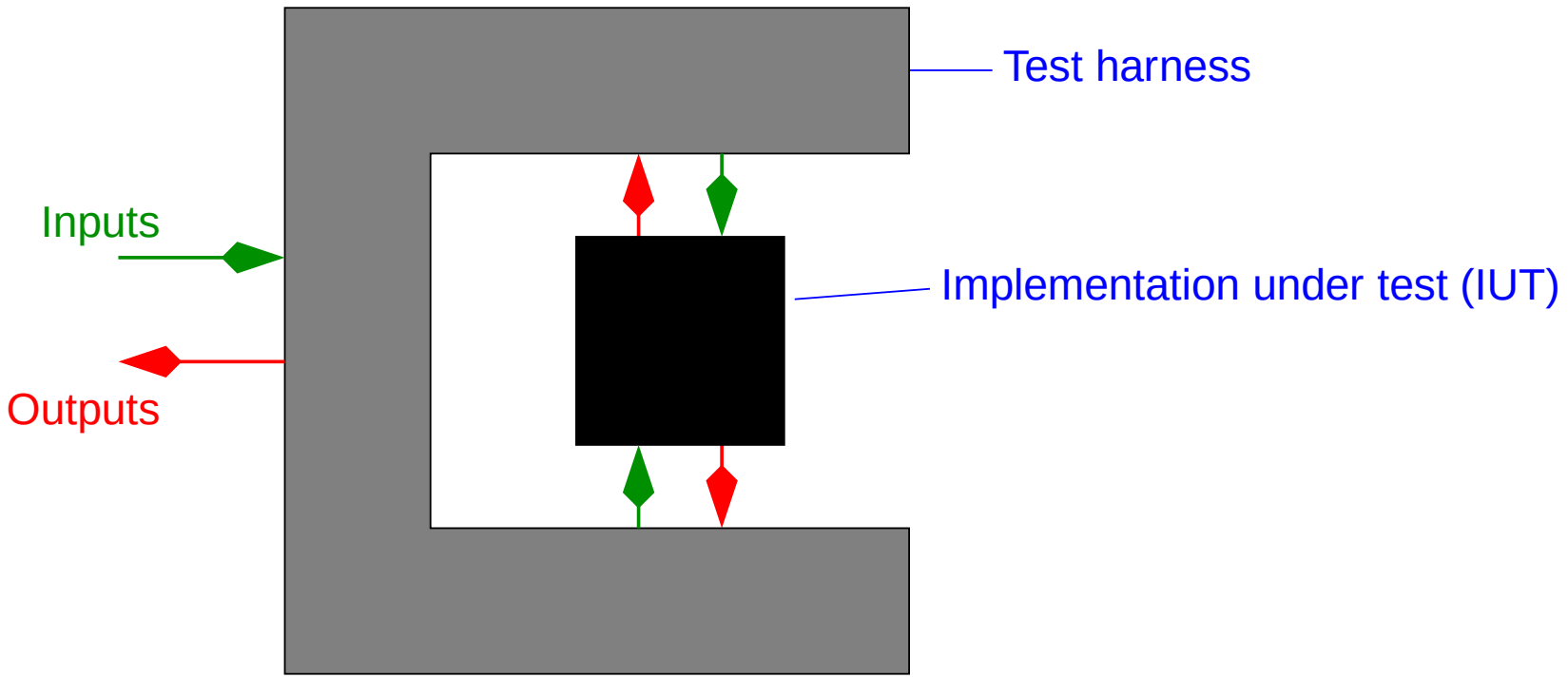
Testing, as traditionally done

The (much simplified) V-Model

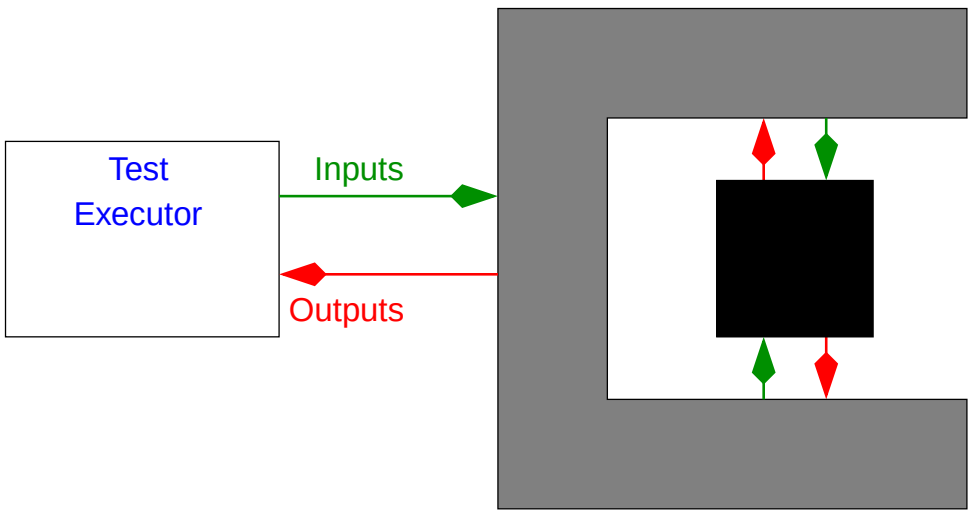


Acceptance Test: testing whether system fulfills end-users requirements

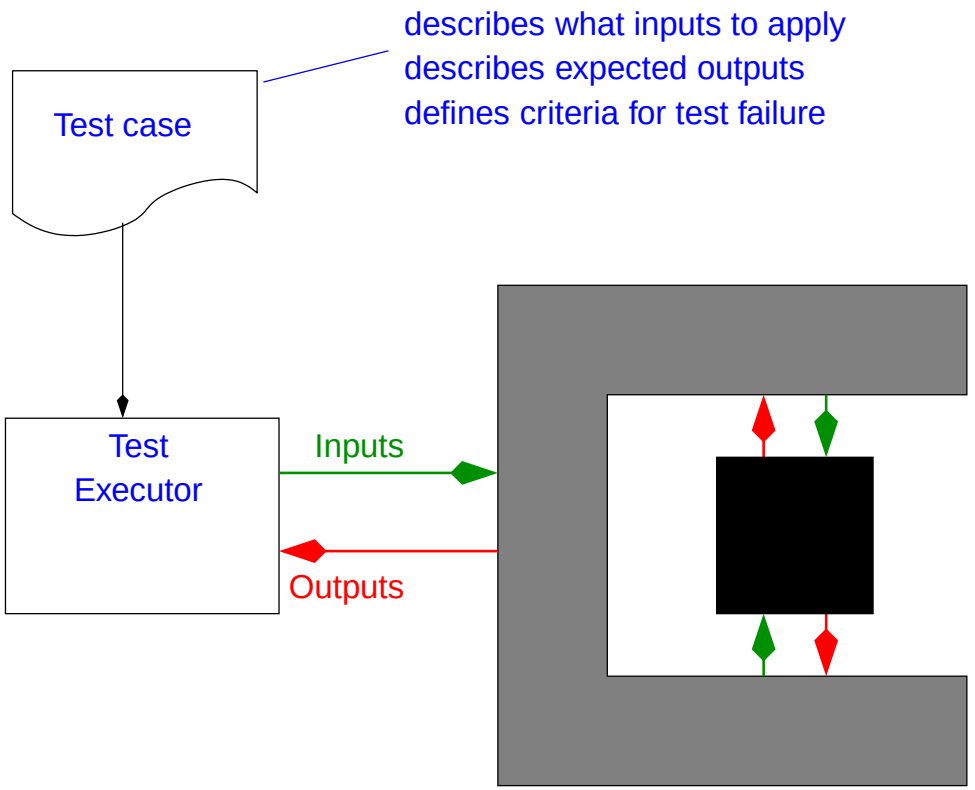
Testing framework



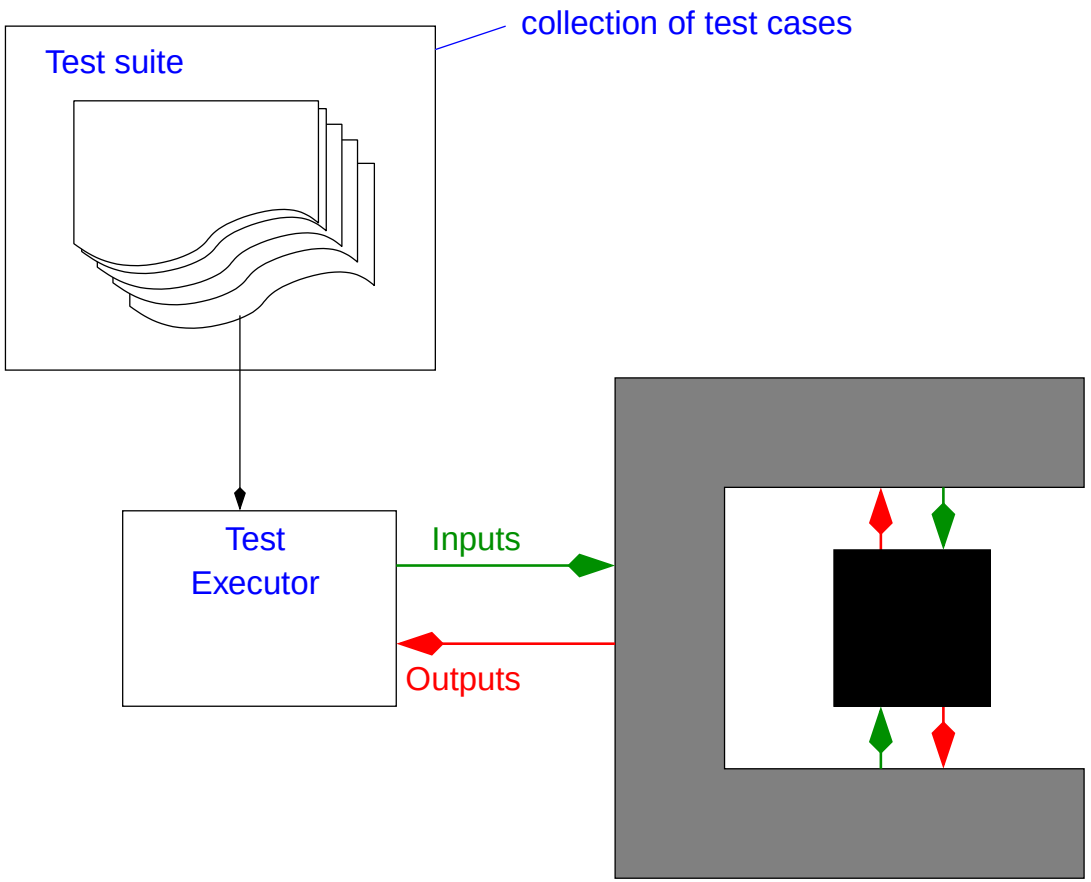
Testing framework



Testing framework

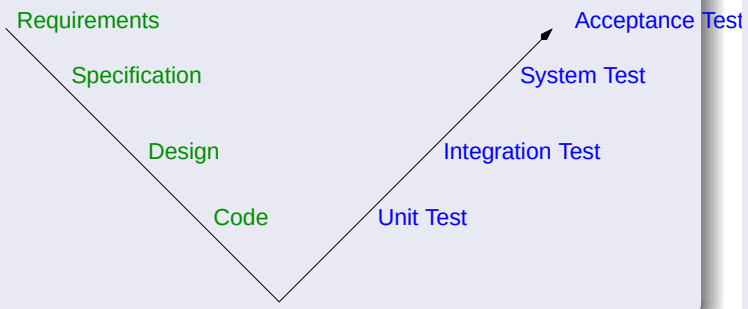


Testing framework



Test cases

V-Model

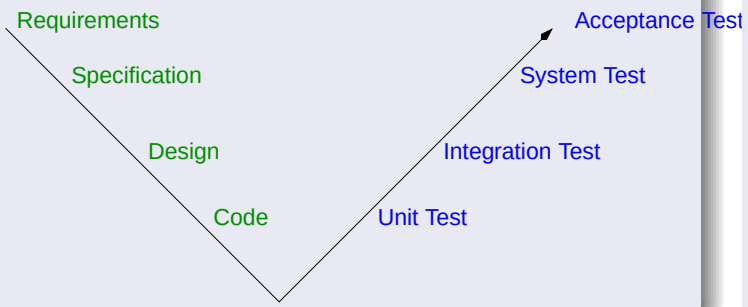


Problems

- test-cases written manually
- writing test-cases time-consuming $\Longrightarrow$ expensive
- “one test-case per requirement” $\Longrightarrow$ not complete
- changes in requirements require changes in test-suites $\Longrightarrow$ continuous effort

Test cases

V-Model



Problems

- test-cases written manually
- writing test-cases time-consuming $\Longrightarrow$ expensive
- "one test-case per requirement" $\Longrightarrow$ not complete
- changes in requirements require changes in test-suites $\Longrightarrow$ continuous effort

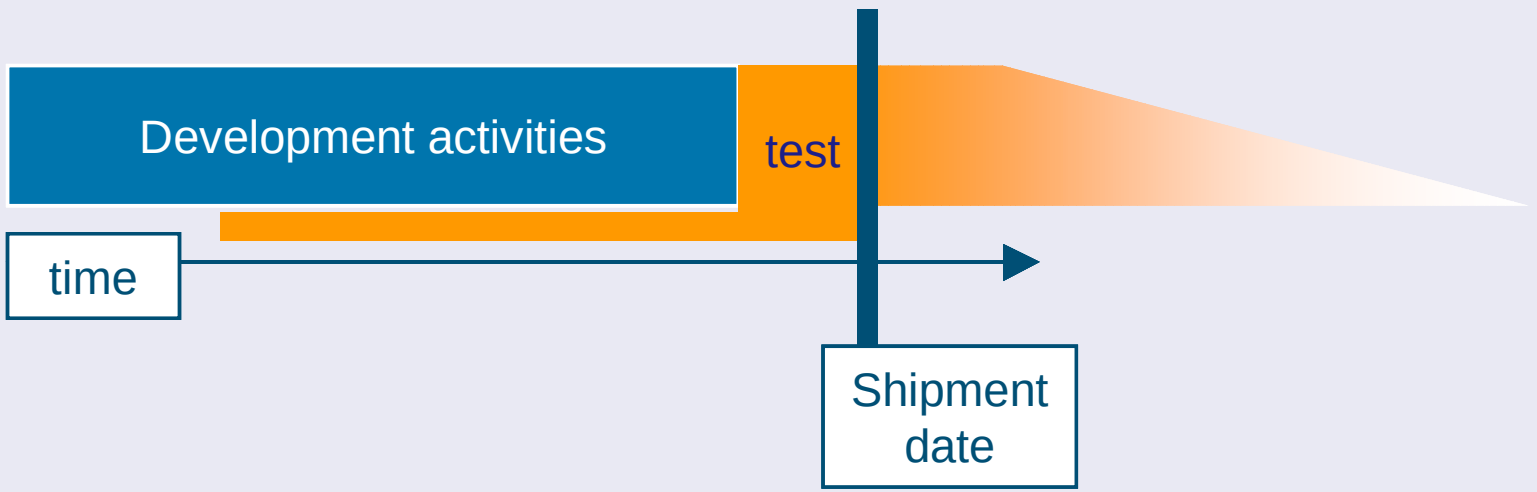
very expensive and time consuming

Economics of Testing

Costs

Testing takes 30% up to 50% of total development cost

Traditional Timeline



Economics of testing

Practical experience

- tendency: testing delayed to the integration phase
- test failures in the complete system not easy to diagnose
- complete testing of the complete system much more time demanding
- testing continues on-site after delivery
- this costs money

Parallel development and testing

- component-testing : saving time by parallelisation
- tight loop between tester and developer
- quick test-case generation
- Automation seems to be a reasonable idea

Economics of testing

Practical experience

- tendency: testing delayed to the integration phase
- test failures in the complete system not easy to diagnose
- complete testing of the complete system much more time demanding
- testing continues on-site after delivery
- this costs money

Parallel development and testing

- component-testing : saving time by parallelisation
- tight loop between tester and developer
- quick test-case generation
- Automation seems to be a reasonable idea

Economics of testing

Practical experience

- tendency: testing delayed to the integration phase
- test failures in the complete system not easy to diagnose
- complete testing of the complete system much more time demanding
- testing continues on-site after delivery
- this costs money

Parallel development and testing

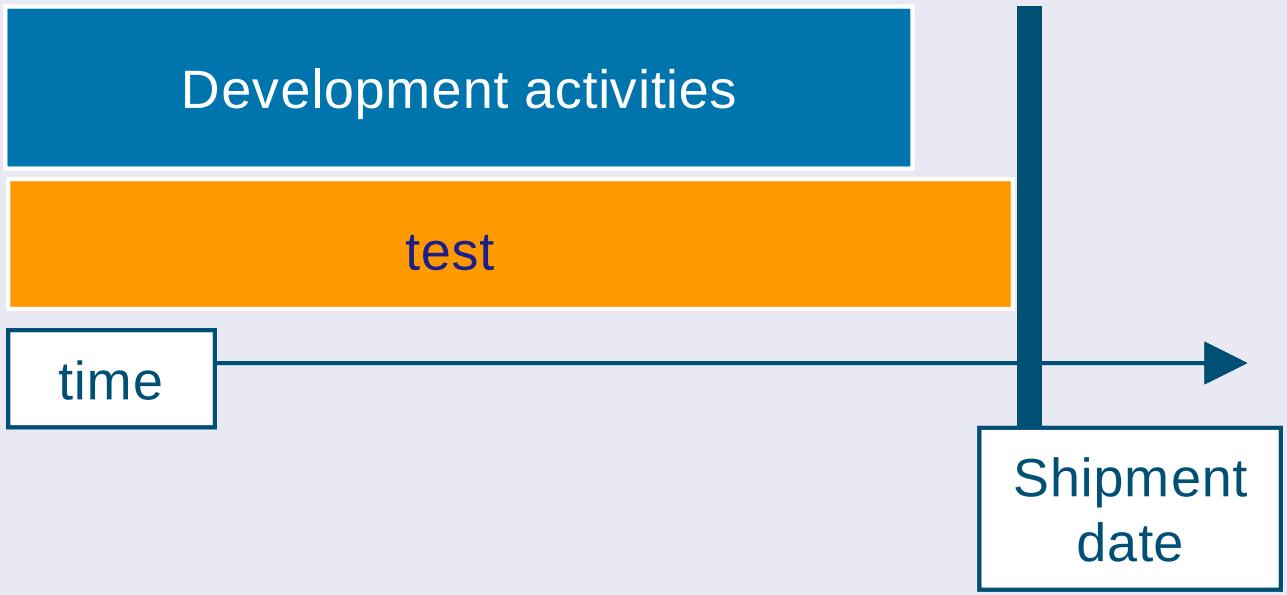
- component-testing : saving time by parallelisation
- tight loop between tester and developer
- quick test-case generation
- Automation seems to be a reasonable idea

Economics of Testing

Costs

Testing takes 30% up to 50% of total development cost

Time: How it should be done



Conclusions

- Test automation very desirable
- Test case generation is the bottleneck

Can Formal Methods help?

Conclusions

- Test automation very desirable
- Test case generation is the bottleneck

Can Formal Methods help?

Conclusions

- Test automation very desirable
- Test case generation is the bottleneck

Can Formal Methods help?

Formal Methods and Testing

Formal methods and software testing have been traditionally seen as rivals.

Dijkstra: Program testing can be used to show the presence of bugs, but never to show their absence!

Why Formal methods for testing?

Formal Methods and Testing

Formal methods and software testing have been traditionally seen as rivals.

Dijkstra: Program testing can be used to show the presence of bugs, but never to show their absence!

Why Formal methods for testing?

Formal Methods and Testing

Formal methods and software testing have been traditionally seen as rivals.

Dijkstra: Program testing can be used to show the presence of bugs, but never to show their absence!

Why Formal methods for testing?

Formal Methods and Testing

"Formal Methods" means: the specification is formal!

Potential Benefits of using formal methods and testing together

Reducing the cost of development by

- Applying testing techniques much earlier in the lifecycle
  $\Longrightarrow$ defects are relatively inexpensive to correct
- Automation of test process
- Generating test cases from the specification
- Generating correct test-executors
- Formally describing the test-case-generation algorithm and proving its correctness

Formal Methods and Testing

"Formal Methods" means: the specification is formal!

Potential Benefits of using formal methods and testing together

Reducing the cost of development by

- Applying testing techniques much earlier in the lifecycle
  $\Longrightarrow$ defects are relatively inexpensive to correct
- Automation of test process
- Generating test cases from the specification
- Generating correct test-executors
- Formally describing the test-case-generation algorithm and proving its correctness

Formal Methods and Testing

"Formal Methods" means: the specification is formal!

Potential Benefits of using formal methods and testing together

Reducing the cost of development by

- Applying testing techniques much earlier in the lifecycle
  $\Longrightarrow$ defects are relatively inexpensive to correct
- Automation of test process
- Generating test cases from the specification
- Generating correct test-executors
- Formally describing the test-case-generation algorithm and proving its correctness

Formal Methods and Testing

"Formal Methods" means: the specification is formal!

Potential Benefits of using formal methods and testing together

Reducing the cost of development by

- Applying testing techniques much earlier in the lifecycle
  $\Longrightarrow$ defects are relatively inexpensive to correct
- Automation of test process
- Generating test cases from the specification
- Generating correct test-executors
- Formally describing the test-case-generation algorithm and proving its correctness

Formal Methods and Testing

"Formal Methods" means: the specification is formal!

Potential Benefits of using formal methods and testing together

Reducing the cost of development by

- Applying testing techniques much earlier in the lifecycle
  $\Longrightarrow$ defects are relatively inexpensive to correct
- Automation of test process
- Generating test cases from the specification
- Generating correct test-executors
- Formally describing the test-case-generation algorithm and proving its correctness

Formal Methods and Testing

"Formal Methods" means: the specification is formal!

Potential Benefits of using formal methods and testing together

Reducing the cost of development by

- Applying testing techniques much earlier in the lifecycle
  $\Longrightarrow$ defects are relatively inexpensive to correct
- Automation of test process
- Generating test cases from the specification
- Generating correct test-executors
- Formally describing the test-case-generation algorithm and proving its correctness

Formal Methods and Testing

Hierons et al. (2009)

It may transpire that its support for test automation is one of the most significant benefits of formal model building.

Formal methods in the design of software systems

Approach

- formal model of the system to be created (specification)
  e.g. process algebra, Petri nets, timed automata . . .
- verify/model check model for design flaws
  using Spin, UPPAAL, MRMC, PRISM, . . .
- implement system
  using your wits.

The process of implementing software introduces errors

Formal methods in the design of software systems

Approach

- formal model of the system to be created (specification)
  e.g. process algebra, Petri nets, timed automata . . .
- verify/model check model for design flaws
  using Spin, UPPAAL, MRMC, PRISM, . . .
- implement system
  using your wits.

The process of implementing software introduces errors

Formal methods in the design of software systems

Approach

- formal model of the system to be created (specification)
  e.g. process algebra, Petri nets, timed automata . . .
- verify/model check model for design flaws
  using Spin, UPPAAL, MRMC, PRISM, . . .
- implement system
  using your wits.

The process of implementing software introduces errors

Formal methods in the design of software systems

Approach

- formal model of the system to be created (specification)
  e.g. process algebra, Petri nets, timed automata . . .
- verify/model check model for design flaws
  using Spin, UPPAAL, MRMC, PRISM, . . .
- implement system
  using your wits.

The process of implementing software introduces errors

Specification-based testing

Idea

Use the specification to do conformance testing

Conformance testing

Use testing to answer:

Is the implementation correct with respect to the specification?

A good reason

if we have a specification anyway, we can well continue using it for testing purposes

Specification-based testing

Idea

Use the specification to do conformance testing

Conformance testing

Use testing to answer:

Is the implementation correct with respect to the specification?

A good reason

if we have a specification anyway, we can well continue using it for testing purposes

Specification-based testing

Idea

Use the specification to do conformance testing

Conformance testing

Use testing to answer:

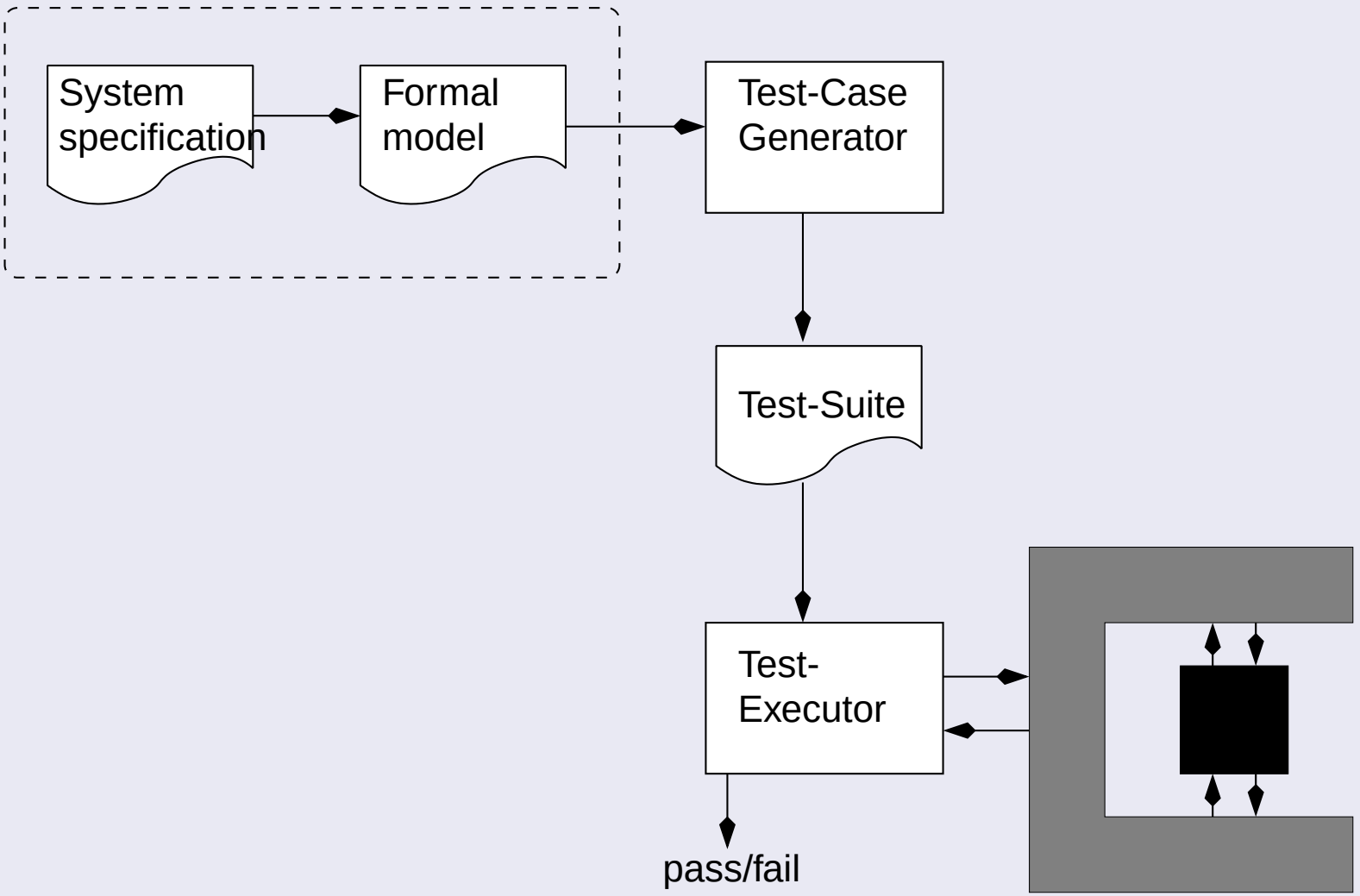
Is the implementation correct with respect to the specification?

A good reason

if we have a specification anyway, we can well continue using it for testing purposes

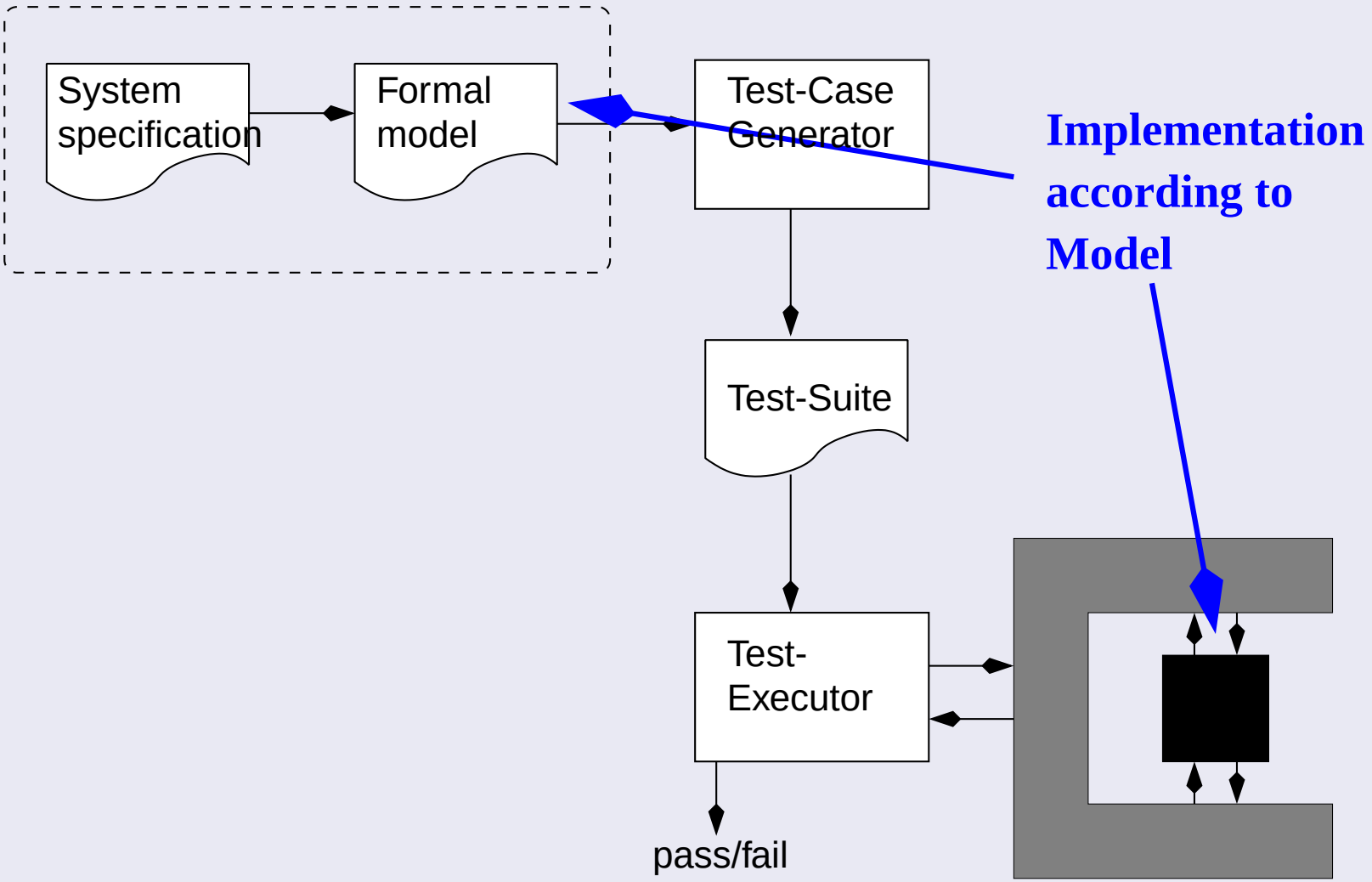
In summary

A new picture



In summary

A new picture



A Formal Approach to Testing

The Ingredients

- a formal behavioral specification $\Longrightarrow$ the intended behaviour
- the test-case generator
  $\Longrightarrow$ a provably correct test-case-gen algorithm
- the automatically generated test-cases
- the test executor
  $\Longrightarrow$ adapting the abstract test-cases to the IUT

A Formal Approach to Testing

The Ingredients

- a formal behavioral specification $\Longrightarrow$ the intended behaviour
- the test-case generator
  $\Longrightarrow$ a provably correct test-case-gen algorithm
- the automatically generated test-cases
- the test executor
  $\Longrightarrow$ adapting the abstract test-cases to the IUT

A Formal Approach to Testing

The Ingredients

- a formal behavioral specification $\Longrightarrow$ the intended behaviour
- the test-case generator
  $\Longrightarrow$ a provably correct test-case-gen algorithm
- the automatically generated test-cases
- the test executor
  $\Longrightarrow$ adapting the abstract test-cases to the IUT

A Formal Approach to Testing

The Ingredients

- a formal behavioral specification $\Longrightarrow$ the intended behaviour
- the test-case generator
  $\Longrightarrow$ a provably correct test-case-gen algorithm
- the automatically generated test-cases
- the test executor
  $\Longrightarrow$ adapting the abstract test-cases to the IUT

A Formal Approach to Testing

How can a test-case generation algorithm be proven correct?

"Formal model" means: we are working with mathematical objects

What we can do

- we can go the whole way: we formalise everything
  specification, implementation, test-case, test-execution
- Then we can reason about the testing approach
- explore its properties
- show correctness of, e.g. the test-case generation algorithm

A Formal Approach to Testing

How can a test-case generation algorithm be proven correct?

"Formal model" means: we are working with mathematical objects

What we can do

- we can go the whole way: we formalise everything
  specification, implementation, test-case, test-execution
- Then we can reason about the testing approach
- explore its properties
- show correctness of, e.g. the test-case generation algorithm

A Formal Approach to Testing

How can a test-case generation algorithm be proven correct?

"Formal model" means: we are working with mathematical objects

What we can do

- we can go the whole way: we formalise everything
  specification, implementation, test-case, test-execution
- Then we can reason about the testing approach
- explore its properties
- show correctness of, e.g. the test-case generation algorithm

A Formal Approach to Testing

How can a test-case generation algorithm be proven correct?

"Formal model" means: we are working with mathematical objects

What we can do

- we can go the whole way: we formalise everything
  specification, implementation, test-case, test-execution
- Then we can reason about the testing approach
- explore its properties
- show correctness of, e.g. the test-case generation algorithm

Pros/Cons of this approach

+ changes in system design $\Longrightarrow$ change in behavioural model
+ recreation of test-cases on button press
+ less error-prone
- formal model needed (needs improvement of Software Engineering methods)
- questionable approach for existing systems

A Formal Approach to Testing

Central Notion: Implementation Relation

- implementation, specification:
  - expressed as Labelled Transition Systems (LTS)
- notions of correctness: implementation relations

$i, s \in \mathrm{LTS}$

$i \leq s \iff i$ is an implementation of s

$\leq \subseteq \mathrm{LTS} \times \mathrm{LTS}$ is an implementation relation

A Formal Approach to Testing

Central Notion: Implementation Relation

- implementation, specification:
  - expressed as Labelled Transition Systems (LTS)
- notions of correctness: implementation relations $i, s \in \mathrm{LTS}$

  $i \leq s \iff$ i is an implementation of s

  $\leq \subseteq \mathrm{LTS} \times \mathrm{LTS}$ is an implementation relation

Testing Hypothesis

Testing Theory

Theory developed solely inside mathematical framework

Practical Testing

concerned with real-life processes, program systems, . . .

far from mathematical

Testing hypotheses

If we can assume, that our implementation can be modelled as an LTS

Then the theorems and properties of our testing theory hold and the test-cases report no false failures

Testing Hypothesis

Testing Theory

Theory developed solely inside mathematical framework

Practical Testing

concerned with real-life processes, program systems, . . .

far from mathematical

Testing hypotheses

If we can assume, that our implementation can be modelled as an LTS

Then the theorems and properties of our testing theory hold and the test-cases report no false failures

Testing Hypothesis

Testing Theory

Theory developed solely inside mathematical framework

Practical Testing

concerned with real-life processes, program systems, . . .

far from mathematical

Testing hypotheses

If we can assume, that our implementation can be modelled as an LTS

Then the theorems and properties of our testing theory hold and the test-cases report no false failures

Testing Hypothesis

Testing Theory

Theory developed solely inside mathematical framework

Practical Testing

concerned with real-life processes, program systems, . . .

far from mathematical

Testing hypotheses

If we can assume, that our implementation can be modelled as an LTS

Then the theorems and properties of our testing theory hold and the test-cases report no false failures

Testing Hypothesis

Testing Theory

Theory developed solely inside mathematical framework

Practical Testing

concerned with real-life processes, program systems, . . .

far from mathematical

Testing hypotheses

If we can assume, that our implementation can be modelled as an LTS

Then the theorems and properties of our testing theory hold

and the test-cases report no false failures

Testing Hypothesis

Testing Theory

Theory developed solely inside mathematical framework

Practical Testing

concerned with real-life processes, program systems, . . .

far from mathematical

Testing hypotheses

If we can assume, that our implementation can be modelled as an LTS

Then the theorems and properties of our testing theory hold and the test-cases report no false failures

This lecture

Part 1: How to specify reactive systems

- Labelled Transition Systems
- A very simple calculus to specify processes

This lecture

Part 2: Distinguishing processes by observation

- Implementation relations, Preorders
- The linear time/branching time spectrum
- Non-determinism
- Observers

This lecture

Part 3: A complete test theory

- Labelled transition systems with inputs/outputs (IOTS)
- Implementation relations for IOTS: ioco
- Test-case generation
- Correctness proofs

This lecture

Part 4: Timed testing

- Timed labelled transition systems (TLTS)
- Implementation relations for TLTS
- Test-cases
- Timed testing using Timed Automata

This lecture

Part 5: ??

Contact

My Coordinates

Henrik Bohnenkamp
E1, Room 4210
Tel 0241 80-21203
e-mail henrik@cs.rwth-aachen.de

In case of questions

- Pop in when you like
- Write an e-mail

Up-to-date information

http://www-i2.informatik.rwth-aachen.de/i2/testing09/