test (14)

Test Generation for Embedded Software - what works, what is needed?

Paul J. Krause
Philips Research Laboratories
&
Surrey University

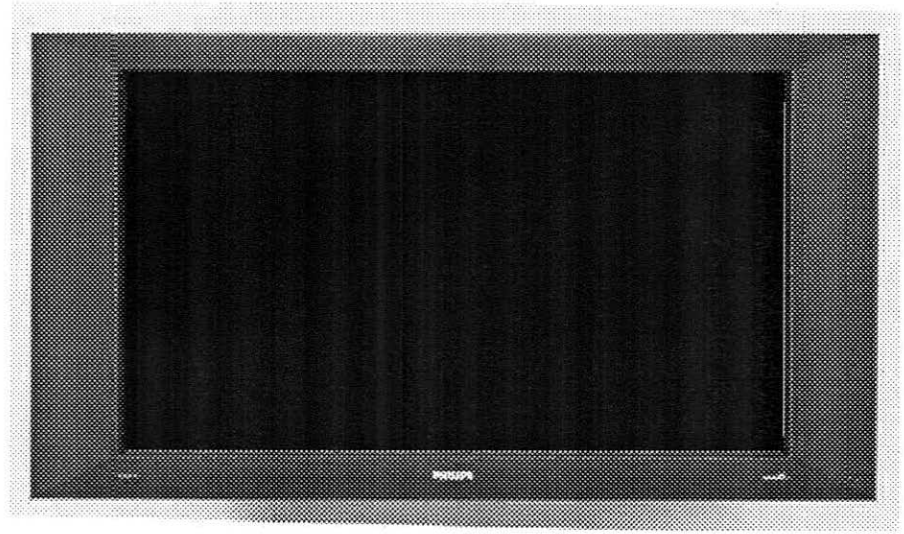
UniS

PHILIPS

Contents

- Software and Philips Electronics
- Automated testing to the rescue
- Two Issues with Test Generation
- The Move to Components
- Conclusions

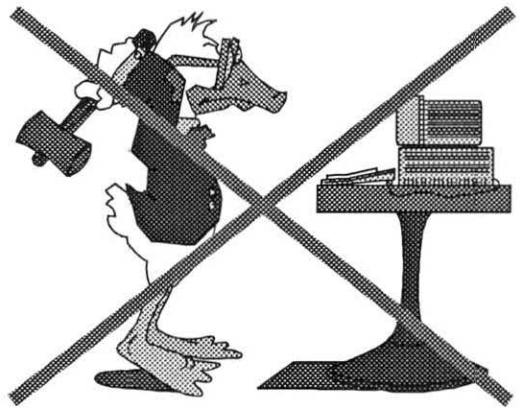
Software - benefits and risks



UniS

PHILIPS

The Drive for Software Quality



Low Defect Rates

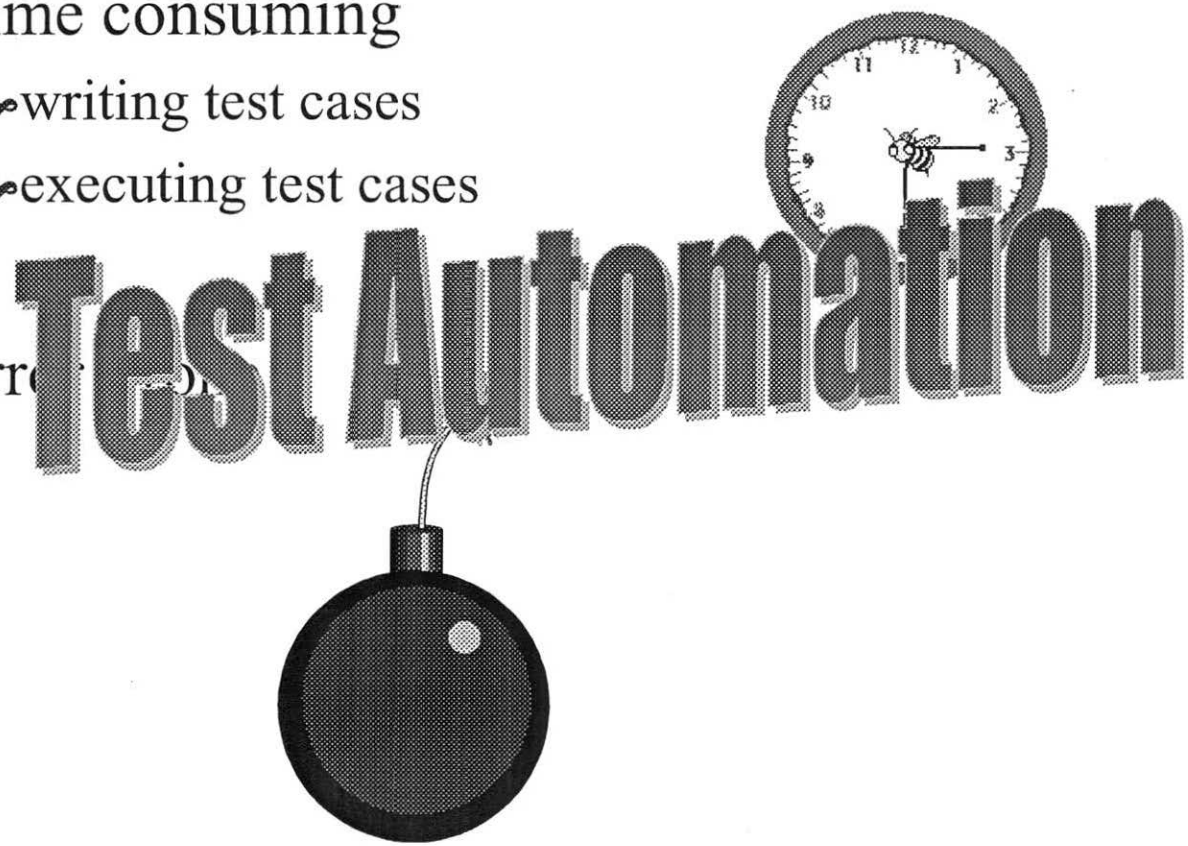
High User
Satisfaction

UniS

PHILIPS

But Testing is:

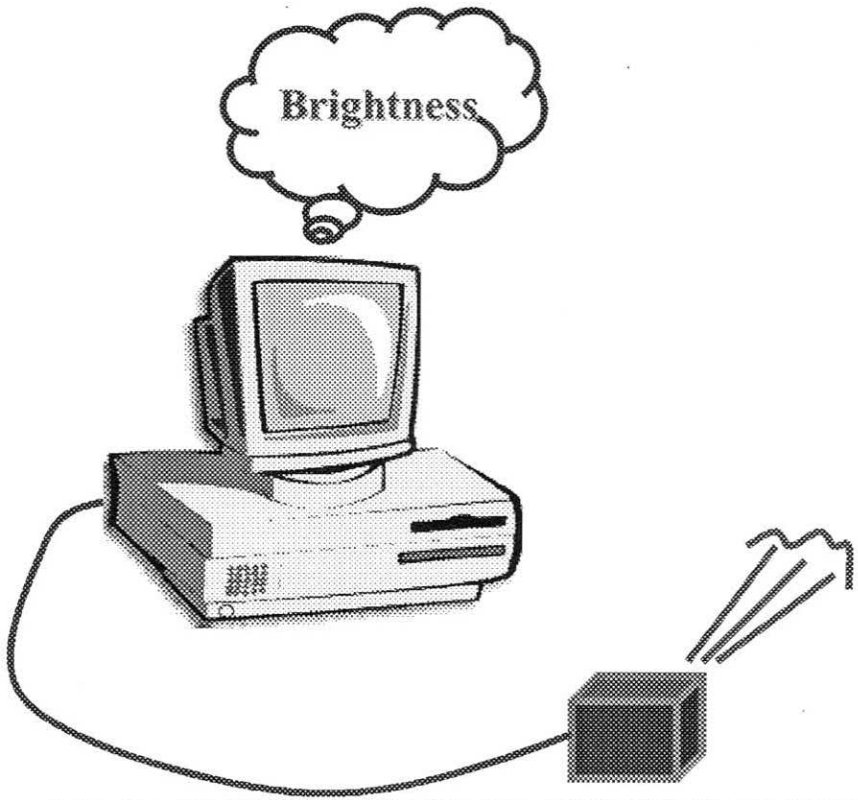
- Time consuming
  - writing test cases
  - executing test cases
- Err[illegible]



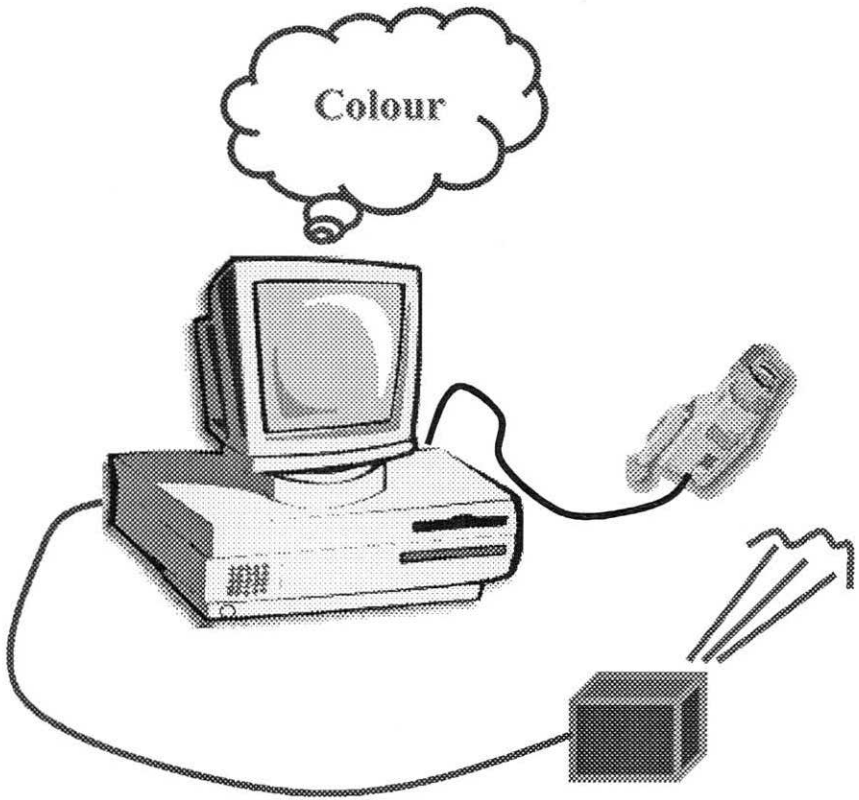
Contents

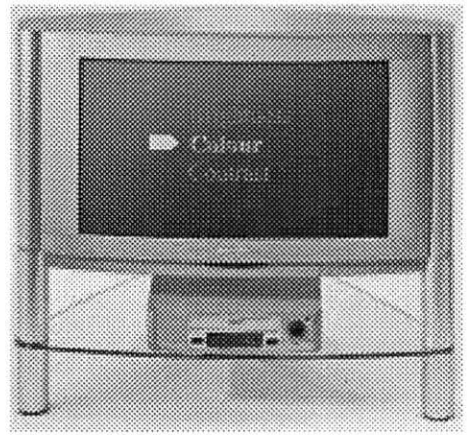
- Software and Philips Electronics
- Automated testing to the rescue
- Two Issues with Test Generation
- The Move to Components
- Conclusions

Test Case Execution:

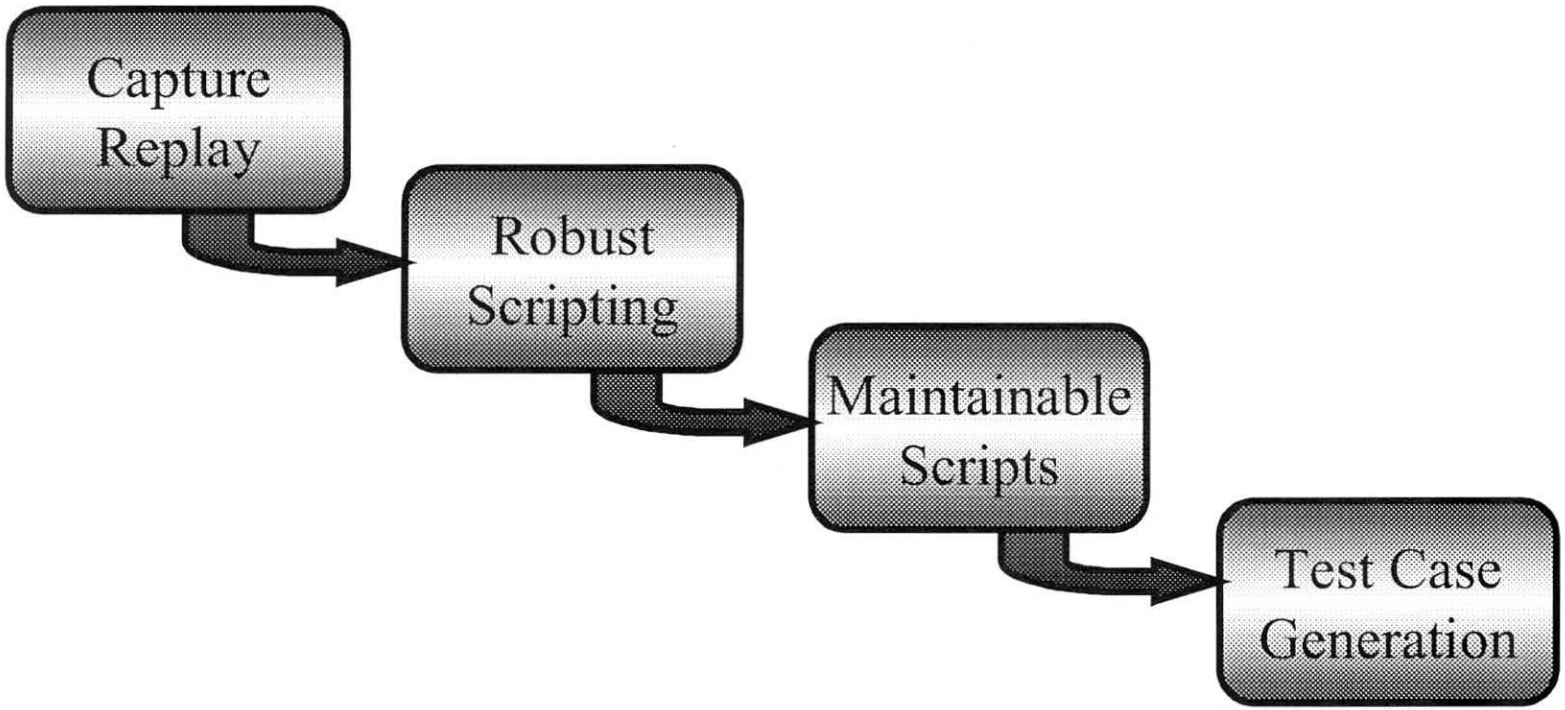


Test Case Execution: Video Feedback





Capture Replay is not Test Automation

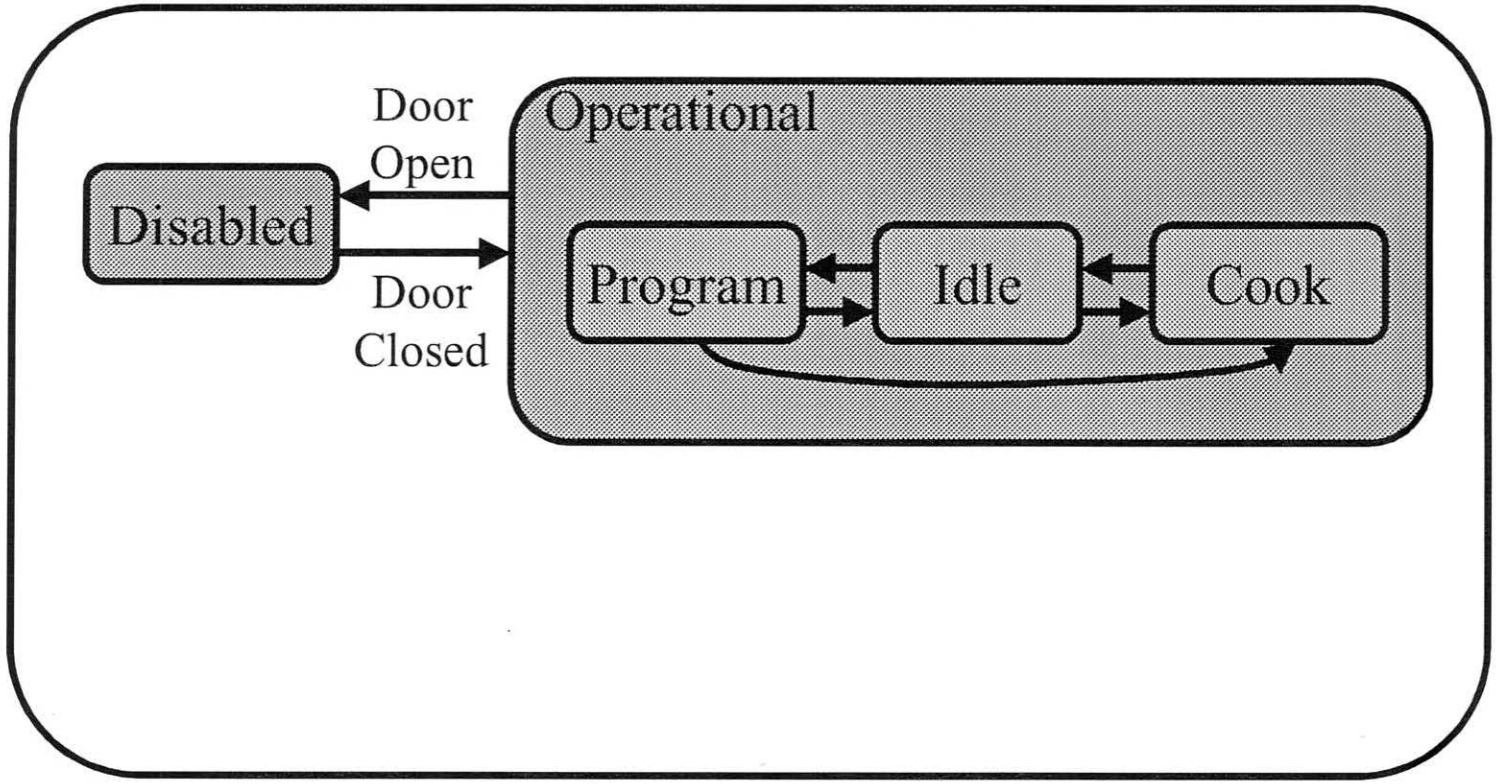


Contents

- Software and Philips Electronics
- Automated testing to the rescue
- Two Issues with Test Generation
- The Move to Components
- Conclusions

Revision - State based module testing

Microwave Oven



Questions

- How to link "abstract" test cases to a specific user interface?

- How to control the state space explosion for system-level test generation?

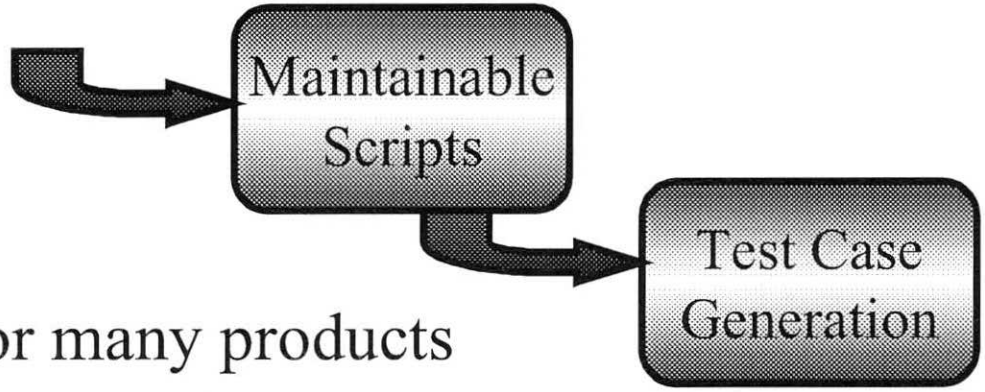
Example state-based tests

Initial Volume	Initial Mute Status	Event	Final Volume	Final Mute Status
Volume < Max	**Sound not Muted**	**Increment Volume**	**Volume Incremented**	**Sound not Muted**
Volume = Max	**Sound not Muted**	**Increment Volume**	**Volume not Incremented**	**Sound not Muted**
Volume < Max	**Sound Muted**	**Increment Volume**	**Volume Incremented**	**Sound not Muted**
Volume = Max	**Sound Muted**	**Increment Volume**	**Volume not Incremented**	**Sound not Muted**

User Actions that Generate Events

- Increment Volume
  - Volume up on Remote Control
  - Cursor right on local keyboard (may vary)
  - Possibly a voice command
- Increment Brightness
  - Sequence of commands on Remote Control (Menu Navigation - may vary across model variants)
  - Combination and Sequence of commands on Local Keyboard

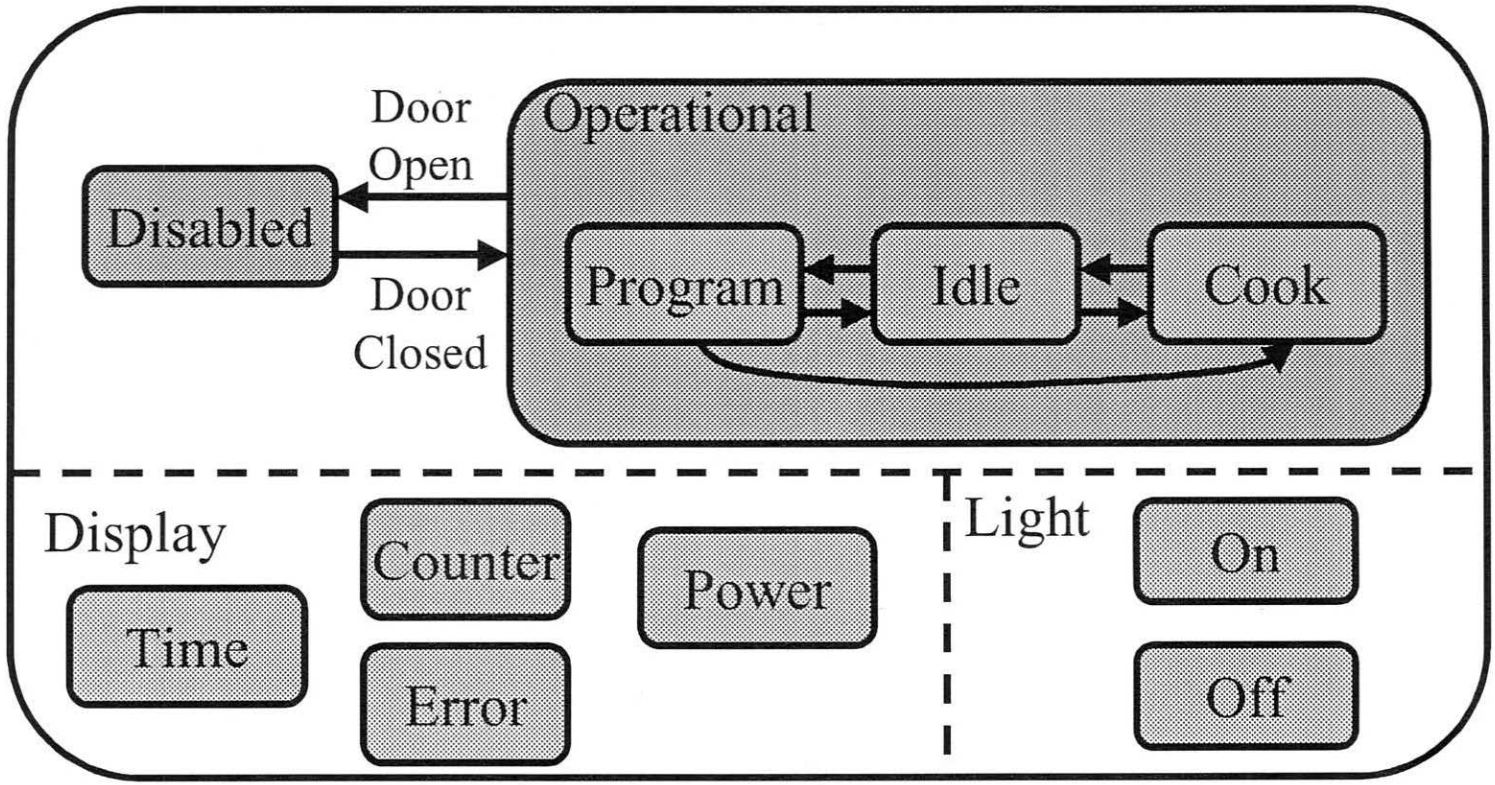
Re-use vs. Re-generation



- For many products
  - the underlying state-behaviour is quite stable
  - User-interface varies as a visible differentiator
- Could use one large model, but
  - does not conform to standard way of working
  - would lead to duplication of common state-behaviour

Revision - State based module testing

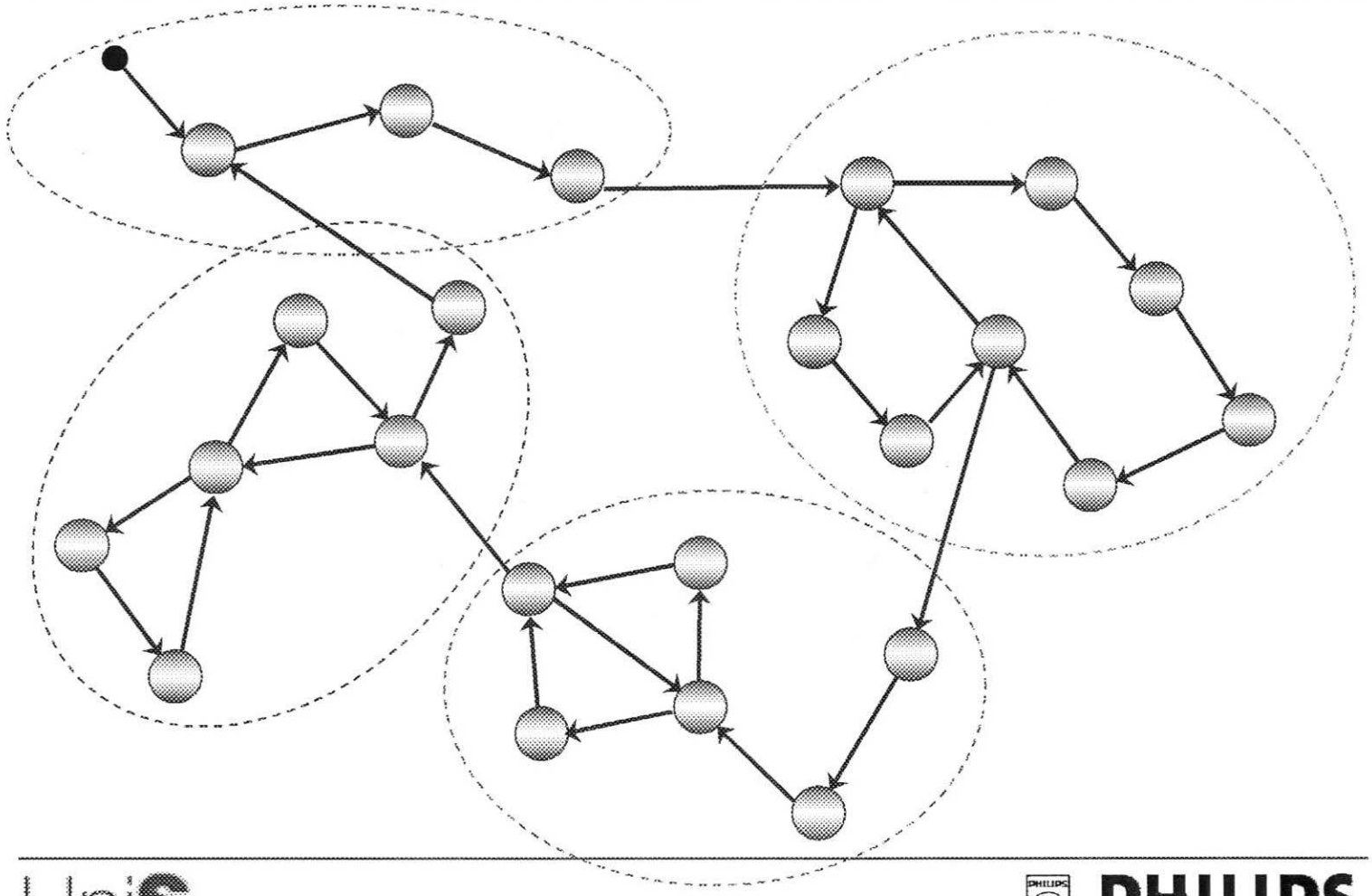
Microwave Oven



How to control state-space explosion?

- Modularise the specification
  - find distinct groups of FSMs, together with the state relations that describe the interactions between those FSMs
- Generate a feasible number of test cases for each module
  - using well founded selection rules
- Integrate these test cases into test suites for the complete system

Big Problem!



Contents

- Software and Philips Electronics
- Automated testing to the rescue
- Two Issues with Test Generation
- The Move to Components
- Conclusions

The Move to Software Components

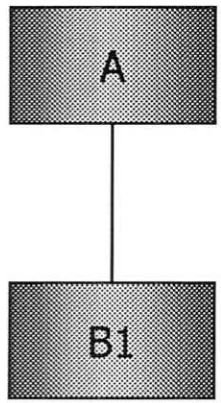
- Consumer Electronics products are members of complex family structures
- Exhibit diversity in:
  - product features
  - user control style
  - supported broadcasting standards
  - hardware technology
- Need to create new products by extending and rearranging elements of existing products

The need for components

- Object-oriented frameworks enable multiple applications to be created from shared *structure* and *code*
  - but changing the structure significantly is difficult
- Component-based approaches let engineers construct multiple configurations with variations in both structure and content
  - component - an encapsulated piece of software with an explicit interface to its environment
  - components - can be used in many different configurations

PROBLEM

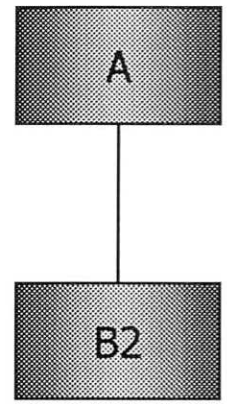
Product 1



Importing B1 into A:

- gives A access to B1
- but puts knowledge of B1 inside A

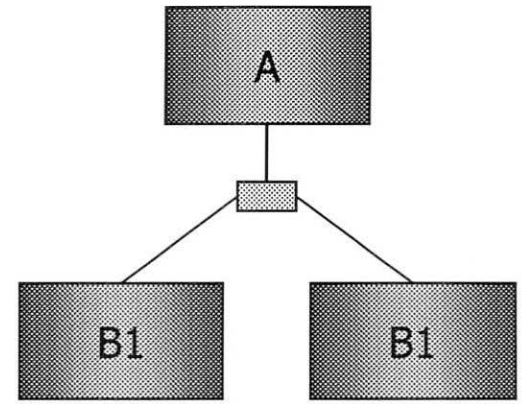
Product 2



So A cannot also combine with B2

SOLUTION

Product 3



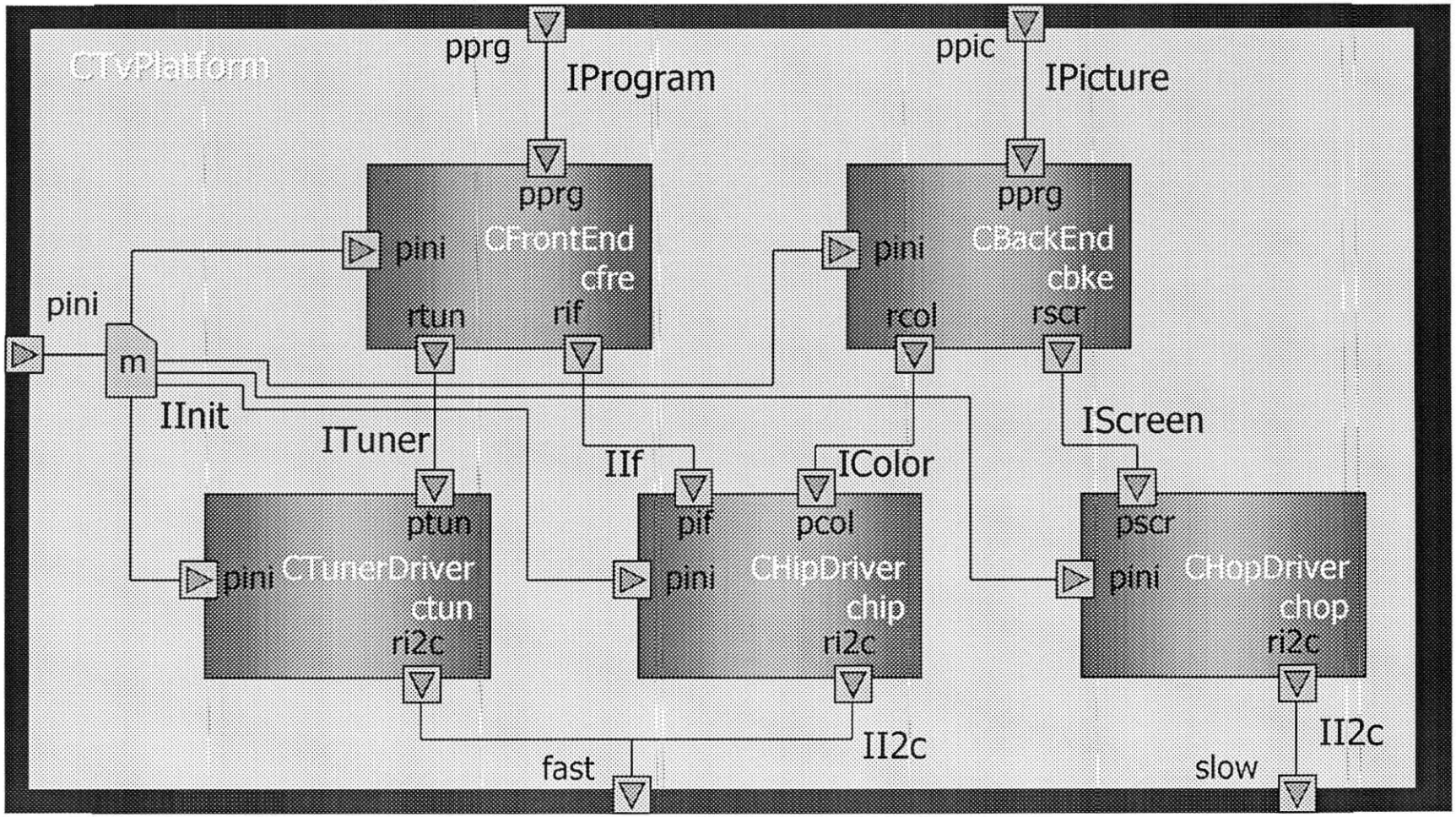
Take binding knowledge out of the components.

- A *requires* an interface of a certain type.
- B1 and B2 *provide* such an interface.
- Binding made at the product level

The Koala Model

- Components
  - units of design development and reuse
- Interfaces
  - a component communicates with its environment through interfaces
  - an interface is a small set of semantically related functions
- A component *provides* functionality through its interfaces
- To do so, it may also *require* functionality through its interfaces

Koala's graphical notation



UniS

PHILIPS

Interface definitions

- Uses a simple Interface Definition Language (IDL) in C syntax. E.g.

```
interface Ituner
{
   void     SetFrequency(int f);
   int      GetFrequency(void);
}
```

Component descriptions

- Describe the boundaries of a component in a Component Description Language

```
component CTunerDriver
{
  provides      ITuner ptun;
                IInit          pini;
  requiresII2c          ri2c;
}
```

Configurations and Compound Components

- A configuration is a set of components connected together to form a product
  - all *requires* interfaces must be bound to precisely one *provides* interface
  - each *provides* interface can be bound to zero or more *requires* interfaces
- It may be useful to compose *Compound Components* from basic components
  - But always, when binding interfaces there must be a unique definition of each function, but a function may be called by many other functions

Emergent behaviour for software components

With Mike Shields, David Pitt and Samantha West at Surrey University

- Need for minimal specifications for interacting components
  - unnecessarily constraining the context in which a component can be used militates against re-use
- Investigate necessary and sufficient conditions to ensure that products developed from compositions of components do not show pathological behaviour

Overview of strategy

- Work at the semantic model level
  - to describe and reason about generic issues related to components and compositions of components
- Develop a simple model of a component
  - what properties of components ensure that the behaviours have "sensible" properties?
  - What are the conditions on pairs of components that guarantee that the "sensible" properties of the individual properties are preserved on their composition?

Background

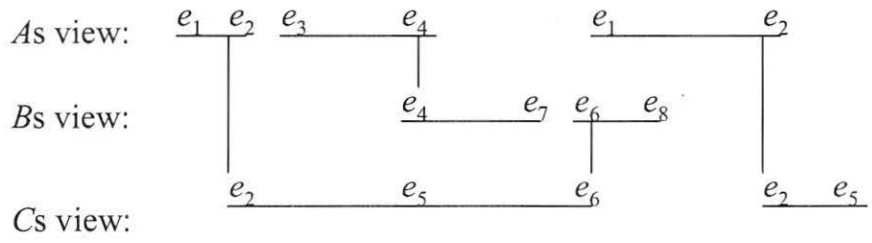
- Dyna… tuples of sequ…
  - ea… res of sig… possible pe…

M.W. Shields,
Semantics of Parallelism,
Springer-Verlag, 1997

- Draws on foundational work by Mike Shields that provides automata with an operational semantics expressive enough to model
  - non-determinacy
  - concurrency
  - simultaneity

Overtaking in Asynchronous Periodic Systems

- Three conditions together are necessary and sufficient to guarantee periodicity
  - coherence - viewers agree on numbers of cycles
  - extensibility - local liveness
  - no overtaking



This behaviour exhibits an *overtaking*. Viewer *A* sees the e_1 period and then the e_3 period. However, Viewer B sees the e_3 period, then the e_1 period.

Contents

- Software and Philips Electronics
- Automated testing to the rescue
- Two Issues with Test Generation
- The Move to Components
- Conclusions

Conclusions

- Test generation from specifications is necessary technology
- There are still some tricky problems to solve
- A more creative use of formal methods may be helpful in support of the move to component-based development