TAMPERE UNIVERSITY OF TECHNOLOGY

Thoughts about model-based test management

Matti Vuori

7.4.2016

Contents 1/2

Contents 2/2

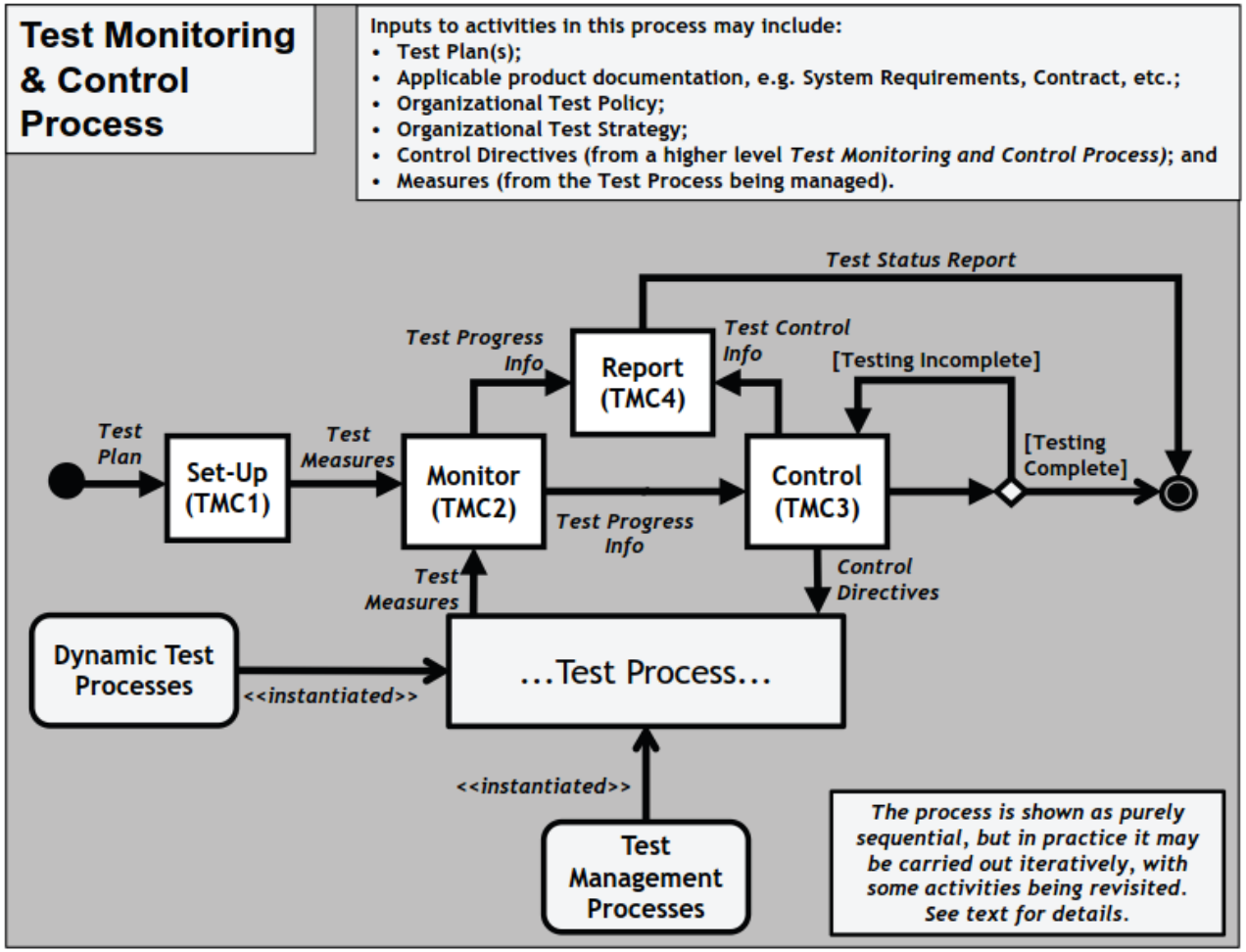
Introduction

- This slide set presents some ideas behind and related to model-based test management
- This is a new paradigm and not fully defined, not to mention understood

Areas of traditional test management

- For reference: Testing process standard ISO 29119-2 defines three areas of test management
  - Test planning
  - Test monitoring and control
  - Test completion
- Of those, monitoring and control is the most essential
- Test planning is not part of test management (just like project planning is not yet project management), but re-planning and refocusing is

Test monitoring and control re ISO 29119



Linked to other activities

- Test management is a process tightly coupled with
  - Product lifecycle management
  - Development management
  - Requirements management
  - Quality management
  - Defect management
  - Build management
  - Release / deployment management
- Quite often these cannot be separated and that is good: it implies that there is an integrated whole (hopefully also a reasonably simple whole)

In practice

- Management of test assets: test repositories and test infrastructures
- Building of test sets
- Monitoring execution of tests – passing, problem areas, test progress
- Making estimates for testing and maturing of product
- Reacting to requirement, design and implementation changes by re-testing
- Adjusting test focus based on needs
- A job for test managers and others

Development management as context 1/2

- Stakeholder management
- Innovation management
- Concept management
- Requirements management
- Progress management
  - Implementations
  - Readiness / releasability
  - Metrics
- Design and implementation infrastructure management
- Build and during-development delivery management
- Test management

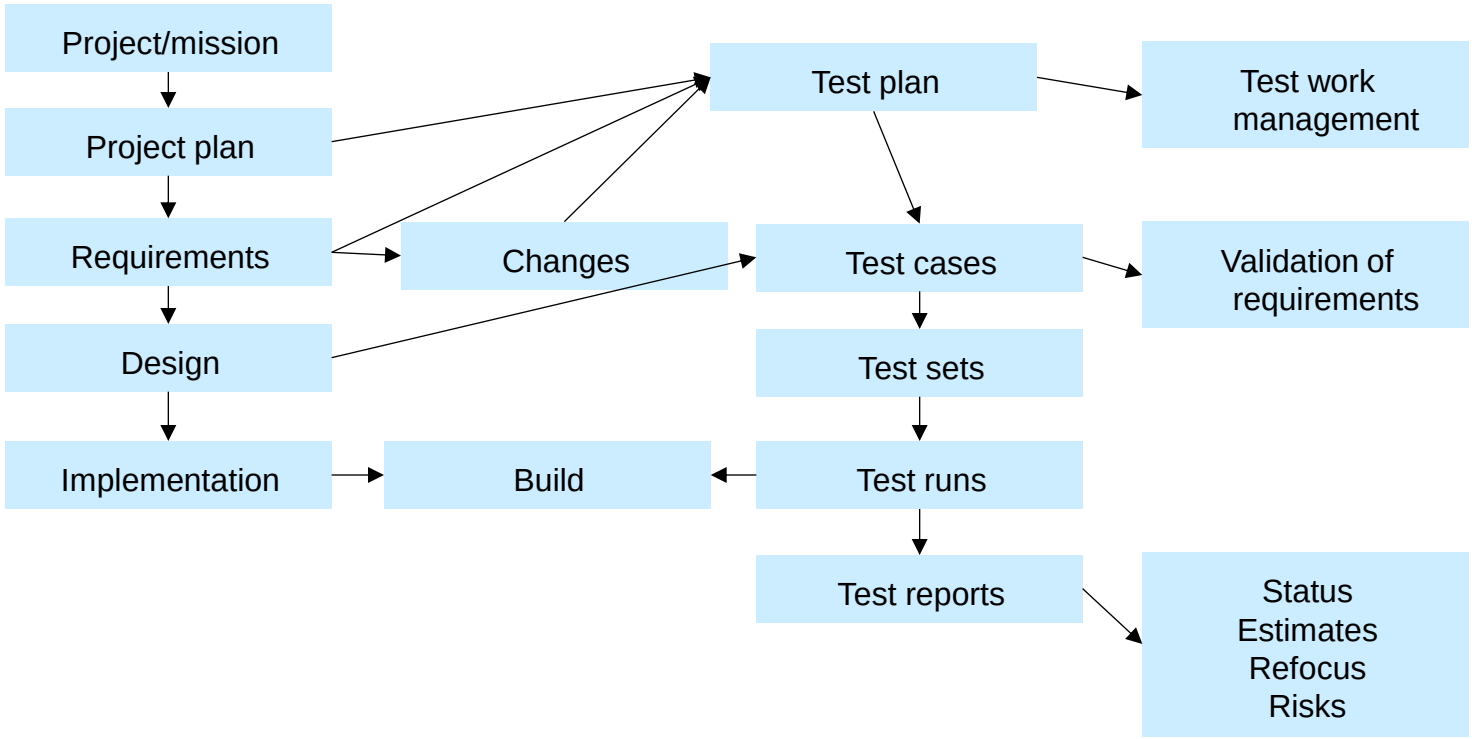
Development management as context 2/2

- Coordination
  - Product (functional) integrity management
- Information management
- Work management
  - Work allocation
  - Distributed work
  - Effectiveness & efficiency
- Quality management
  - External quality / value management
  - Debt management
- Risk management
- Personnel management
  - Motivation, roles

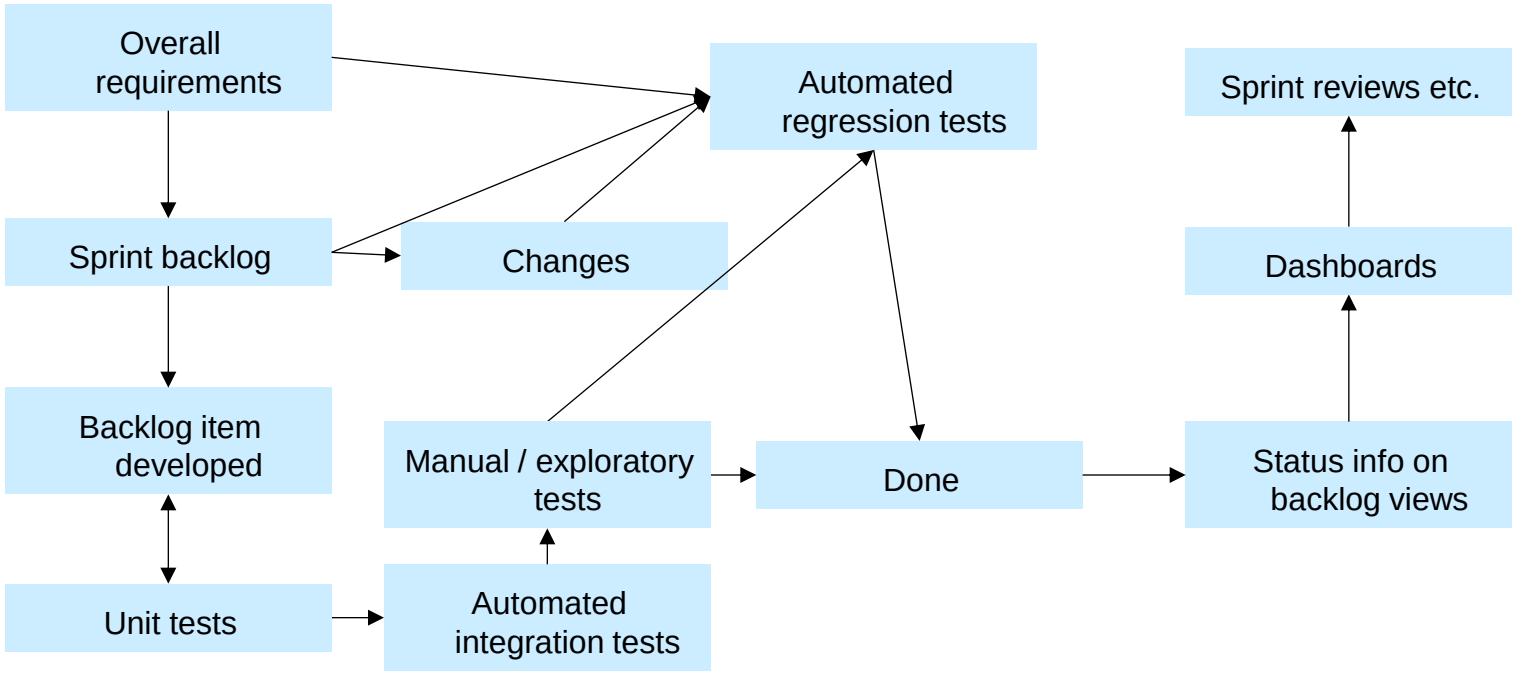
Basic ideas of traditional test management

- Centralised on the whole set of tests and their status
- Development flow traced two ways with testing (traceability matrix)
- Verification and validation of implementation -> by tracing V&V of requirements
- Requirements coverage essential
- Changes in requirements invalidate tests
- New build invalidates everything -> regression testing
- Monitoring involves status, coverage, estimates, work done – metrics tell what's happening

Traditional test management flow (simplified)



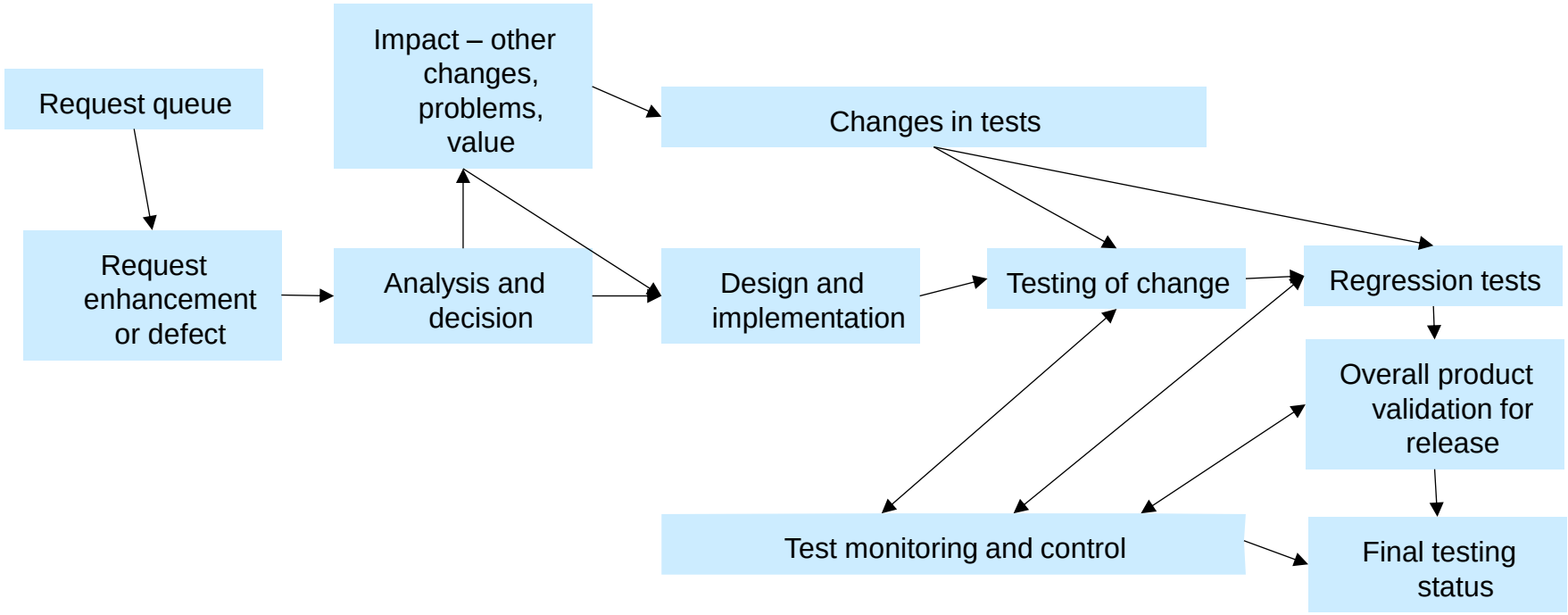
Test management flow in agile (simplified)



Basic ideas of agile test management

- Focuses on high level on status of backlog items (done?)
- Coverage monitoring on low level
- Blends on traditional views as companies grow in maturity

Test management flow in maintenance mode (simplified)



Test management at various test levels 1/2

- Test management matters most at levels where management is needed
  - System testing & system integration testing (systems of systems integration)
  - Putting together parts from different teams and contractors
- Team-level / local CI (especially unit testing) can often be just simple development management
- CI is a tool for the mental process of test management

Test management at various test levels 1/2

- In any context one needs to select what details leave out
  - To reduce complexity in information management
  - To focus on the essentials
  - To let freedom for local contexts to manage their work as they feel free (depending on the amount of trust…)

What could model-based test management be? 1/2

- Based on models (!)
  - Model of the system under development
    - Behavioural model that represents what the system should be able to do
  - Model of the development flow
    - Abstract flow, not tying teams or units
  - Model of information flow
    - Development bases that influence testing; that have changes that invalidate tests
  - NOT: Model-based testing in traditional sense is not required, but linking of testing to other models

What could model-based test management be? 2/2

- Monitoring and control
  - Testing of activity elements (such as some interaction)
  - Status of activity elements
    - Readiness state: Specified -> designed / reviewable -> implemented / testable -> tested [at various levels]
- Models include triggers for possible action
  - Specified -> do test preliminary design
  - Designed -> do review, update test models
  - Implemented -> run tests with updated models

Other models utilised 1/2

- Dependency models for design information
  - Rich, not only simplistic dependency links
  - Type of dependency
  - Rules
  - Associated actions for changes
  - Includes standards for critical development (safety, security)
- Impact models
  - How any proposed changes impact design, implementation, testing, validation

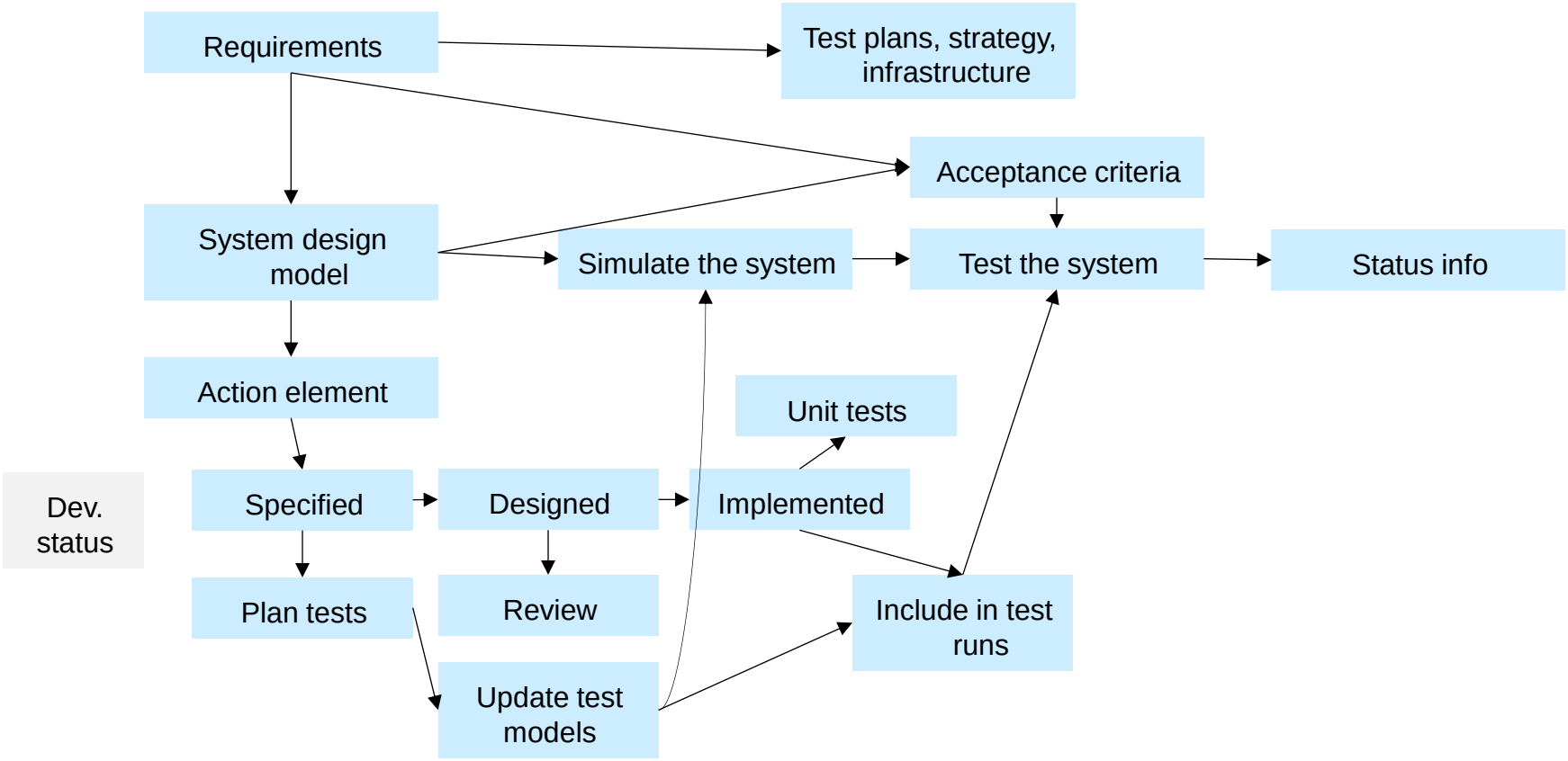
Other models utilised 2/2

- Customer, business, process models
  - Customer's criteria, priorities and preferences
  - Business processes
  - Work processes
  - Ecosystem processes

From links and metadata to model elements

- Traditional test management connects items together as links or metadata
- They are used for tracing and as flags for showing need to redo or revalidate things
- In the model-based approach items can have active and automatic role in the test configurations, test designs, control of test execution and evaluation of test results

Model-based test management flow (simplified)

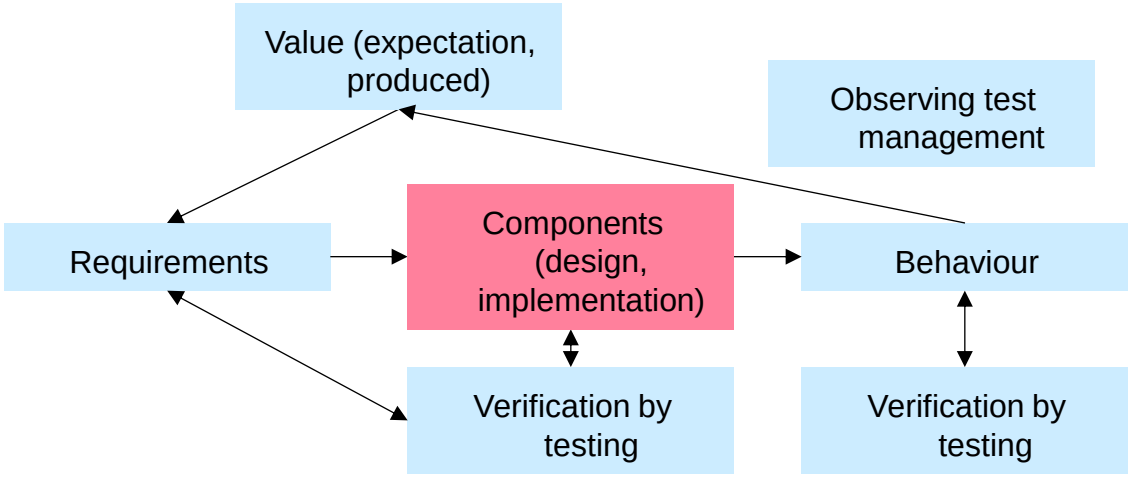


Good test basis

- When the development basis is included as elements of the overall model, it is also readily available for verification and validation
- Traditionally this has been assumed due to tracing things back to requirements, but there have been difficulties due to the parts of development flow having different concepts, thus requiring mental mappings (= guessing)
- When system behaviour is defined in an clear way it can form a good basis for at least functional acceptance criteria

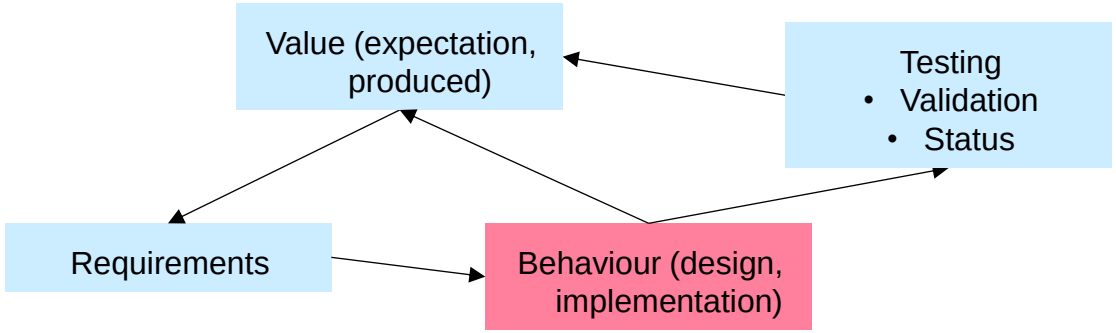
Focus change and streamlining in test management

TRADITIONAL



MODEL-BASED

Key to this: working on abstract design level, not implementation level



From "management" to collaboration and learning

- Test management is traditionally "management", external monitoring and tool for using power
- Redefining it to a collaboration system:
  - Visibility
  - Status of shared and linked tasks
  - Reduce graphs and add information about enabled activity, needs and focus areas of action
  - Shared knowledge creation and learning
- Dynamic testing-related information can help teams be effective and break fixed linear work patterns

Answers to more productive questions

- Traditional test management answers questions such as "what is the status of testing?", "what have been done?", "what is the general progress of testing?"
- Model-based test management could give answers to better, more productive questions
  - "What could we test soon"
  - "What can we test now?"
- This is real status information that matters

Fix the traditional BIG problem

- Traditional problem is that two system elements work, but don't work together. There may be solid tests and test management for both.
  - Without working together, nothing has been achieved.
- So, let's focus on how the interaction works. Drop excuses. Focus tests on the interaction and monitor how it progresses. Force teams to work for the whole.
  - You get what you measure! Don't want "tested components", but tested system behaviour

Loopback: from metrics to mark-up...

- Feedback from traditional test management is usually in the form of metrics.
- Model-based test management could give its information as "mark-up": marking of system elements in a repository with information about whether they should work
- That way, system simulations and emulations can directly use the test results and promote real understanding of how the system currently behaves

...and even removal of features...

- Of course the feedback can also actually block some immature behaviour in actual releases
- The model-based whole can make the blocking easier than traditional approaches by just using some simple variables at abstract level, instead of touching implementation code
- The idea of blocking some functionality is obviously not new, but doing it from the "test interface" might not be common
- (Note though, that the test models are not the same as the design models so this needs planning)

Change management -> changes in tests

- Model-basedness can force consideration of changes in requirements or designs: "Injected" change will simply change behaviour
- Similarly it will automatically affect test designs if the tests are model-based (use test models or are configured from any design model element)
- This is a critical benefit for development large-scale requirement products or when there is a lot of changes in requirements
- The potential can be untapped with holistic modelling (not by low level behaviour modelling)

Benefits 1/4

- Straightforward, effective view to development and testing
  - Oriented towards system activity and how building it proceeds – the highest level of "engineering"
  - Bypasses structural views – they offer no value (don't care about components or code – just what they do)
  - Supports more directly model-based testing, but does not require it

Benefits 2/4

- Binds viewpoints
  - System activity is the common denominator for architects, designers, different teams and testers – basing management on that forces communication that is clear and focused
  - Suits feature-oriented development better that structural views
- Expandable
  - Design-level models are expandable to higher level models of systems of systems, customer, usage profiles etc.

Benefits 3/4

- Supports learning about the system
  - Behaviour-focus lets people learn by managed testing how the system really works
  - Enables improvements and re-thinking of the system
- Enables innovation
  - Abstract view on technology hides details, is portable to new technologies
  - Eases rapid changes in platforms, implementation technology

Benefits 4/4

- Automatable
  - System is automatable – entering a state starts action
  - No manual scanning and interpreting of complex development information in daily work
  - Less possibility to neglect, to forget, to not notice

Pitfalls

- Must not get stuck on just one view to system, but support richness of developing
- Triggers should not mandate action, but enable it
  - No automatic starting of workflows that involve humans
  - Instead, full, clear knowledge / information support and tools for initiating the actions (such as infrastructure virtualisation and reservations for test environments)

What kinds of testing?

- When development is based on interaction models, it is natural to do model-based testing
- But at system level, all companies do other styles of testing too: traditional scripts, exploratory testing; data based testing they all fit in
- "-ility" tests need management too – usability, security, performance…
- Unit testing is outside the scope of test management; developers should do it as integrated part of their work