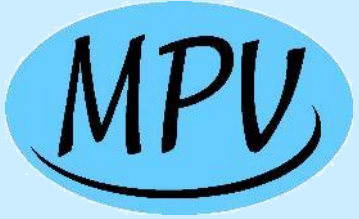



# On various testing topics: Integration, large systems, shifting to left, current test ideas, DevOps

Matti Vuori www.mattivuori.net matti.vuori@mattivuori.net @Matti_Vuori

# TUT lecture series of SW Technologies:
# Integration and testing

Matti Vuori

# Contents

# Integration is valuable from many viewpoints 1/2

- Creation purpose
  - Combine things, features together to form a news whole
  - Creating a new version for testing, deployment, publishing
- Symbolic purpose
  - Process that shows achievement for everyone
- Organisational purpose
  - Central point of combining work

# Integration is valuable from many viewpoints 2/2

- Measuring purpose
  - Integration measures progress
  - Showing areas that keep up or don't
- Assessment purpose
  - Integration testing allows for trying the system
  - Creation of a whole to understand it
- Problem identification
  - Point to identify and solve problems

# At many levels

- Typically integration has many levels
  - Software level: Module, feature, component
  - Hardware / software integration
  - Product integration: software with overall product / platform, product configurations
  - System integration: systems of systems
- Need to start the upper levels too as soon as possible
  - Issue is related to scaling of agile development

# Various situations

- Building a brand new systems
  - Building direction?
  - Top to bottom? Need stubs, allows assessment of concept
  - Bottom to top? Needs drivers, builds a robust platform, but the concept…
  - => Customer / business needs + development needs
- Request, new requirement
  - Continuous development flow… just add a piece

# Requires

- Strategy and rules
- Discipline
- Good distributed infrastructure and tools
- Configuration management – product/customer configurations, versions; product and environments
- Good test design
- Work in progress management
- Synchronisation
- Prioritisation
- Sharing information
  - What to prepare for, what should work and be tested

# Good characteristics for integration testing system

- Fast to set up
- Reliable
- Handles load
- Supports testing
- Visibility
- Fast feedback to developers
- Integrated into workflows
- Supports distributed development

# Things to test

- Functional tests at many levels
- Security tests
- Simple performance tests
- Static code analysis
- Code metrics
- Adherence to architecture
- …

# Integration test issues 1/2

- Need to think of goals
  - Just testing for a “working” build for system tests? (As in building a house with working plumbing for testing it)
  - Doing real verification, proper finding of defects? System testing?
  - Learning about the system?
- Real integration tests
  - Not just repetition of lower level (unit) tests
- Quality over quantity
  - Good, relevant tests essential

# Integration test issues 2/2

- Target: functioning real system
  - When something is integrated to a real system, progress is made
  - Implementation is worth nothing without working integrated whole
- How to provide fast feedback?
  - Test run design – last things first, regression tests separately
- No bottlenecks
  - Simple, efficient flow

# Continuous integration mindset

- Test everything automatable right away
- Test automation works best in very small batches
- Don't waste time and effort switching environments, just use CI
- Learn to test everything fast

# Continuous integration pathologies

- Speed overrides quality of tests
- If tests take long, they are simplified or not run at all
- Psychological need to keep radiators green – design that way
- Environment standardised, less testing of configurations
- No system testing, no manual testing
- CI as magic tool

# Speed of testing

- Optimisation of tests based on dependencies
  - Automatic optimisation?
- Parallel execution of a test set
- Many test sets
  - Parallel execution with different target times

# Common problems 1/2

- Development management
  - What to test, what should be working?
- Change management
  - APIs removed or behaviour has changed
  - Unplanned changes
  - Planned changes that have negative effect – regresssion
- Implementation
  - Wrong use of APIs (parameters, values)
  - Implementation does not match designs
  - Fragile components

# Common problems 2/2

- Test environments
  - Hardware or simulators not available
  - Cannot do "pre-integration"
- Test design
  - Slow tests – feedback from integration takes time
- Bad testability
  - Cannot test system integration easily without a fully working environment

# Success factors in integration testing 1/2

- Common configuration
  - Operations, development, testing, QA
  - No surprises
- Knowing what to test – sharing information
  - What should be working and what not?
  - Preparation for what is coming
- Working on good testability
  - Technology choices, architecture, design for testability
  - Testability review
- Good documentation of APIs and such and API discipline

# Success factors in integration testing 2/2

- Testing small as small as possible pieces at a time
  - Incremental integration & tests
- Select integration order – development order that supports goals
- Real integration tests
  - Not just repetition of lower level (unit) tests
- Pre-integration at lower level
  - As in developers doing local builds
- Minimise feedback delay to developers
  - Test run design – last things first, regression tests separately

# TUT lecture series of SW Technologies:
# Testing of large systems

Matti Vuori

# Contents

# Some concrete problems of testing large systems

- Knowledge of how it works.
- Test environments.
- Configurations.
- Changes.
- Plenty of different technologies.
- End to end testing.
- **Large is usually at the same time complex and business critical – difficult equation.**

# Problems are of course related to goals…

- Creating new systems, new value fast.
- Being more agile.
- Working with new business partners.
- Succeeding with less people.
- Managing risks.
- Having robust technical quality while complexity increases.
- Creating great customer and user experience.

# Keywords for success

- Shared understanding, goals, priorities.
- Collaboration.
- Shared configurations.
- Work in progress control.
- Test management.

- …No silver bullets…

# Changing test focus

- Modules -> Overall system, business.
- Functionality -> Cyber security, customer experience.
- Technology management -> Complexity, diversity management.
- What is implemented -> What brings value (testing in feature teams).

# Some solutions: Organisational

- Are the main challenges technical or human?
- DevOps
  - Collaboration / teams.
  - Developers, systems management, QA working with common information, goals, systems.
  - Open up personal, silent knowledge to information systems, databases, teams. No heroes.
  - Supported with overall automation, database-derived configurations, sharing information.

# Some solutions: Test approach

- Integration approach
  - Start integration very early.
  - One party responsible for integration.
  - Emphasis on high-level test.
  - Raising inspection focus to integration.
- Changes managed globally with configuration control.
  - Database-derived configuration
  - No surprises in testing, operations, from manual fixups.
- Status views – what should be working, what not?

# Some solutions: Testability

- Testability is critical.
- Design for testability.
- Testability reviews by other teams.
- “Technology agnostic tools” such as Robot Framework
  - As small adaptation surface as possible

# Some solutions: Fast feedback

- Effectiveness vs. efficiency
- Continuous top level integration tests
  - Test selection for focus, priorities, risks.
  - Test optimisation for speed, rapid feedback. Smaller tests, faster.
  - Exploratory testing is the fastest.
- Changes in implementation?
  - Technology-agnostic test design (Robot Framework) with small adaptation layer.

# Some solutions: Understanding the system & users

- Keyword-driven, high level tests.
  - Robot Framework already in wide use.
- Using tests to understand the system.
- Using tools to reveal dependencies, relations.
- Usability & user experience testing & analysis for understanding users & customers.
- Bring customer and usage information closer to all developers.
- From component development to features – meaning, purpose for work, basis for verification & validation.

# Some solutions: Understanding system in production

- A/B testing in production?
  - Two alternative variations deployed.
  - Collecting data for comparison.
- Big Data from production.
  - Customer, user profiling.
  - Reality-based priorities.

# TUT lecture series of SW Technologies:
# Shifting testing to left

Matti Vuori

10.11.2016

# Shifting testing to left?

- Response to doing testing at the end of processes – even in agile
- Left means two things
  - In process flow
  - In time
- The sooner testing is done
  - The shorter the feedback loop
  - The simpler and cheaper the fixing of defects
  - The more solid the platform for development and testing
- **Could you do more of it?**

# Examples 1/3

- Development practices
  - Feature teams that test features as fully as possible before product integration and QA
- Evolving continuous integration
  - Doing security and stress testing in continuous integration platform instead of a separate environment
  - Doing as much as possible of system testing and end-to-end testing in CI platform
  - Hardware testing in CI

# Examples 2/3

- Simulation
  - Testing with behavioural test models (Model-Based Testing) before implementation
  - Using simulators before real environments are available
- Analysis
  - Programmatic analysis for code, architecture before testing
  - Doing reviews

# Examples 3/3

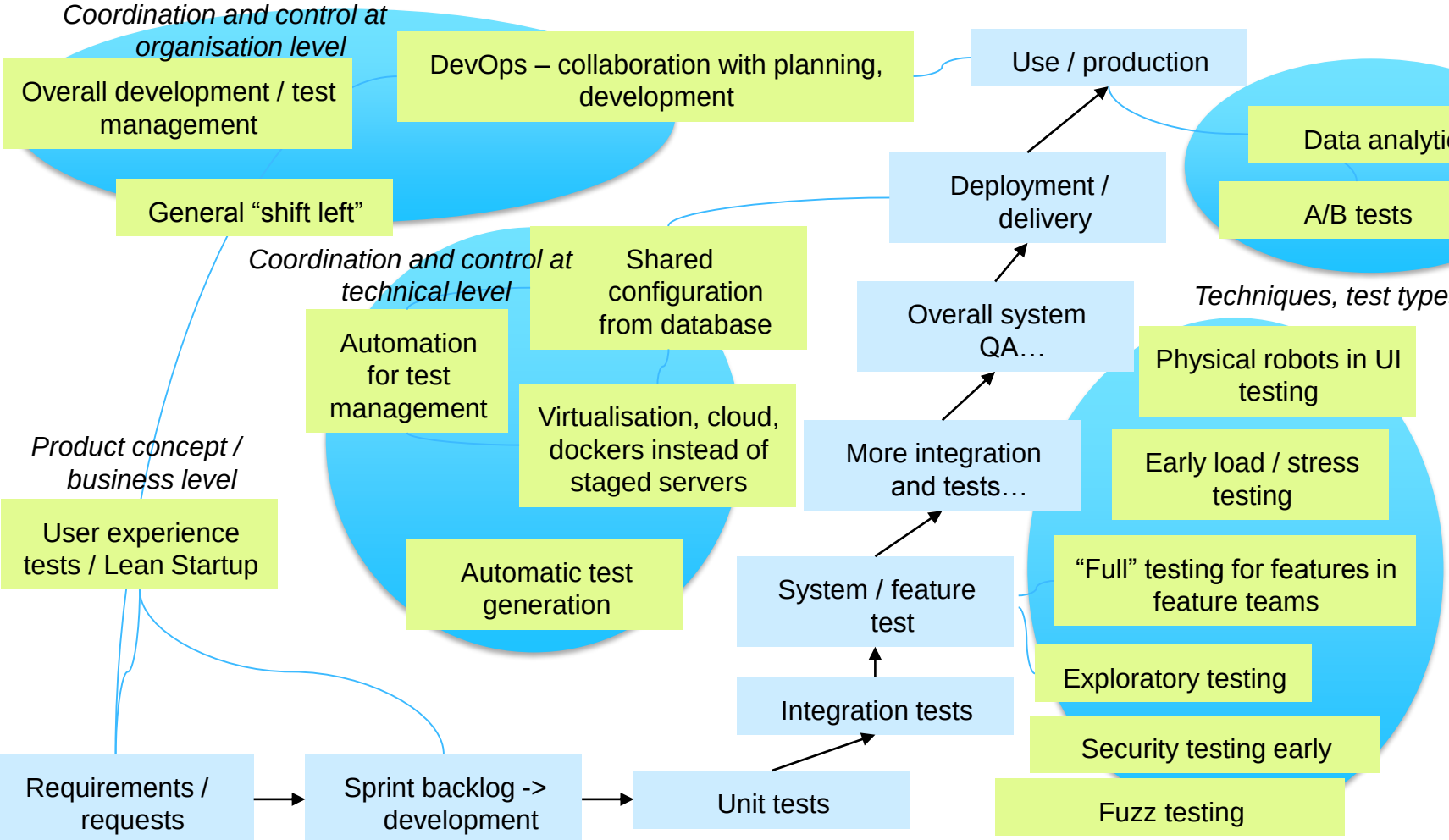
- Rapid testing
  - Doing exploratory testing immediately after something is implemented, not waiting for any test code to be written
  - Helps in creating automated test also
- Prototypes
  - Low-fidelity prototypes for user interface testing
- In general, the old rule: start testing as soon as possible

# TUT lecture series of SW Technologies:
# Some current testing ideas

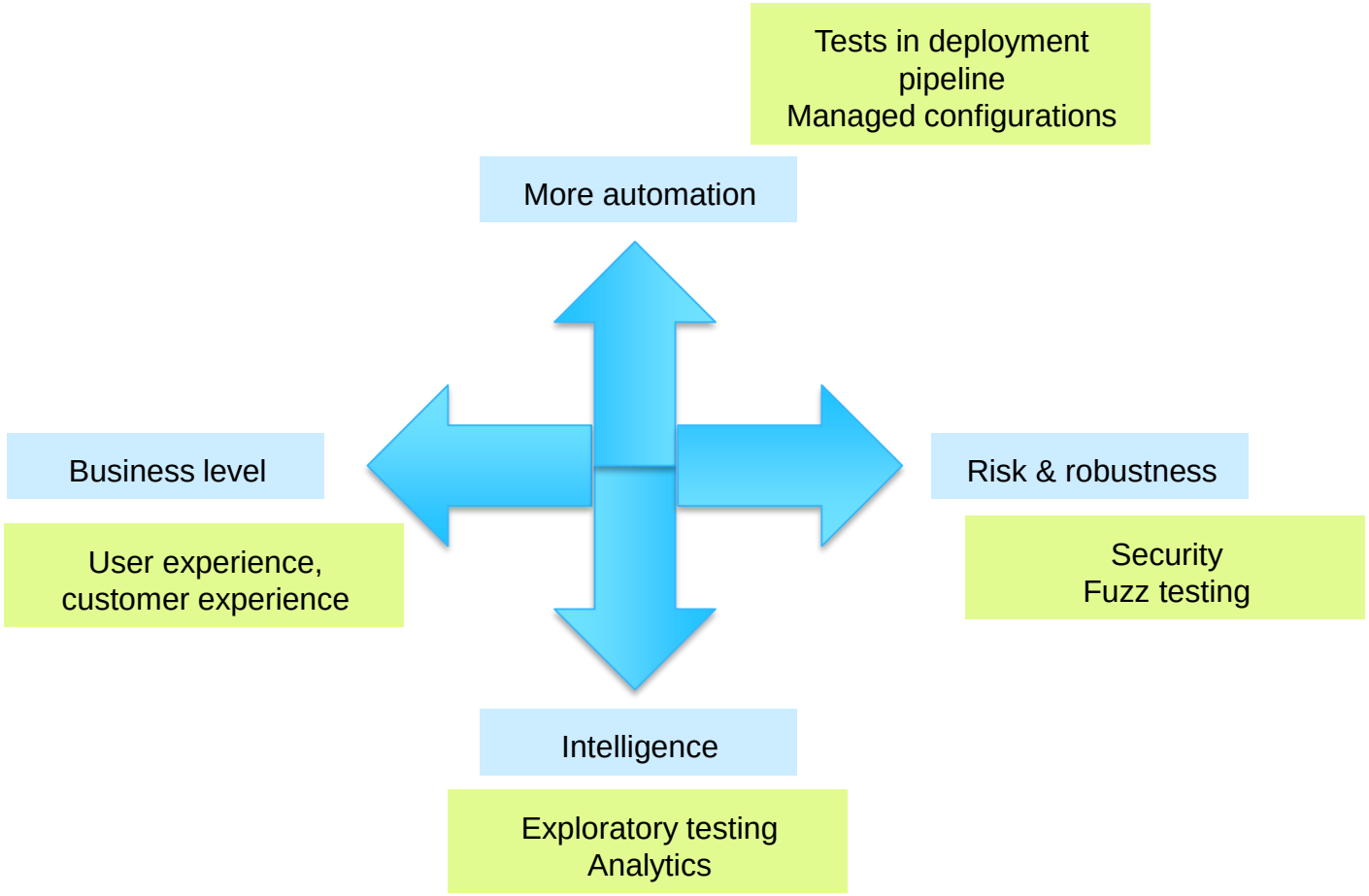
Matti Vuori

12.12.2016

# Some current testing ideas



# Movements


Tests in deployment pipeline
Managed configurations
More automation
Business level
User experience, customer experience
Risk & robustness
Security
Fuzz testing
Intelligence
Exploratory testing
Analytics

# TUT lecture series of SW Technologies:
# About DevOps and testing

Matti Vuori

# Contents

# What is DevOps

- Wikipedia says:

**DevOps** (a clipped compound of development and operations) is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably

https://en.wikipedia.org/wiki/DevOps

# Some views to it

- It is a philosophy
- Practice where every unit, team, works for same goal in a synchronised way
- Optimisation of the whole s/w production pipeline, not local optimisations
- Just good professional collaboration?
- Some see it as a methodology
- Automation everywhere – testing, deployment, configurations
- Extending continuous delivery to operations side
- **Beware: Different people see DevOps in very different ways**

# It’s about collaboration

- Between operations, development, QA
- In planning projects, products, in designing, implementing and hosting & maintaining them
- Specifying environments
- Ensuring testability for everywhere
- Selecting common tools and systems
- Doing problem solving, diagnostics
- Creating teams for those tasks

# It’s about sharing

- Information – what do we together need to put out next? Work on that decision, not something else

⇒ Prioritisation, control of work in progress

- What is supposed to work and what not?

=> Clear targets for testing

- Configurations, environments – everyone working on/against the same

=> Global configuration control, environments created from database without manual work

⇒ Testing environments match the production "exactly"

⇒ Less surprises

# DevOps is part of progress

| Area | Examples |
| --- | --- |
| Technical progress | Daily integration -> Continuous integration -> Continuous deployment -> DevOps |
| Organisational progress | Less siloes -> co-operation -> real collaboration and sharing |
| Agility progress | Programmatically created environments robust platform, fast to deploy |
| | |
| | |

# Managing work in progress

- What production needs, can digest
  - Pull from operations
- Everyone focuses on doing that well
- Small increments just as in agile, continuous deployment…

⇒ Simplicity, focus in testing

⇒ Collaboration

# System things it uses

- Configuration management -> shared configurations
- Environments built from configuration information
  - Never any manual configuration
  - Virtual machines and Dockers obviously help here
- Development management and test management
  - Shared view to what is happening, progressing, what comes next
  - Total pipeline approach – from requests to deployment and usage (ideas from continuous deployment)

# Related important competences

- Configuration management
- Business understanding
  - Sharing the goals
- Process understanding
  - Avoiding "local optimising"
- Collaboration, coordination and communication skills
  - Orientation towards sharing
- Automation skills
  - Test automation
  - Configuration management
  - Test environment management

# "Global" repository as tool for collaboration 1/2

- DevOps suggest this idea (orig. from Jari Lehto, Nokia)
- System information
  - Configurations – dev, test, production
- Customers and users
  - Priorities, preferences, tailoring
  - Utilisation profiles
- Development rules – standards etc.
- Development status
  - What is going on, next steps, what should work
- Testing status

# "Global" repository as tool for collaboration 2/2

- Data (big and small) available for everything
- Opportunities for intelligence
- => Empowering people
- => Away from blindness
- => Understanding goals
- => Focusing, optimising data sets and designs against them

# Possible data in more detail

| Class | Characteristics | Role in test management |
|---|---|---|
| **Customer information** | Customer profile, needs, systems, usage profile, priorities | Drive test design |
| **Requests** | Flow, external, critical to respond | Require tests, test evaluation basis |
| **Requirements** | Flow, changing, internal | Require tests, affect test creation / generation, test evaluation basis |
| **Designs** | Changing, internal, have status (idea… testable), expressed as models | Require tests, affect test creation / generation, tests depend on status of |
| **External requirements / standards etc.** | Changing, critical | Require tests, affect test environments and tests |
| **Test models** | Changing, respond to designs and requirements (i & e) | Source for test generation |
| **Tests** | Changing, respond to designs and requirements (i & e), will be executed | Main source of monitoring, need planning, timing, coordination |
| **Test environments** | Changing, respond to customer profiles | Need planning, designing, testing, coordination of use of |
| **Deployment configurations** | Changing, respond to customer profiles, configuration models | Need testing, drive test configurations, depend on development status |
| **Monitoring data** | Flow, external, produce operational models | Affect test design |

# Perhaps silliness in DevOps talk…

- "DevOps engineer" sought in job adverts
  - Why is such an "occupation" needed? Isn’t collaboration just professional practice and necessary today?
- "DevOps tester"…
- Seeing DevOps as strict, defined methodology
  - People like methodologies, opportunity for consultants, certificates.
  - Every company needs to find their way of DevOps that fits their business and culture