COMPUTER ENGINEERING SERIES

Advanced Testing of Systems-of-Systems 2

Practical Aspects

Bernard Homès

ISTE

WILEY

Advanced Testing of Systems-of-Systems 2

Advanced Testing of Systems-of-Systems 2

Practical Aspects

Bernard Homès

ISTE

WILEY

First published 2022 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Control Number: 2022944148

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-78630-750-7

Contents

Dedication and Acknowledgments

Inspired by a dedication from Boris Beizer[1], I dedicate these two books to many very bad projects on software and systems-of-systems development where I had the opportunity to – for a short time – act as a consultant. These taught me multiple lessons on difficulties that these books try and identify and led me to realize the need for this book. Their failure could have been prevented; may they rest in peace.

I would also like to thank the many managers and colleagues I had the privilege of meeting during my career. Some, too few, understood that quality is really everyone's business. We will lay a modest shroud over the others.

Finally, paraphrasing Isaac Newton, If I was able to reach this level of knowledge, it is thanks to all the giants that were before me and on the shoulders of which I could position myself. Among these giants, I would like to mention (in alphabetical order) James Bach, Boris Beizer, Rex Black, Frederic Brooks, Hans Buwalda, Ross Collard, Elfriede Dustin, Avner Engel, Tom Gilb, Eliahu Goldratt, Dorothy Graham, Capers Jones, Paul Jorgensen, Cem Kaner, Brian Marick, Edward Miller, John Musa, Glenford Myers, Bret Pettichord, Johanna Rothman, Gerald Weinberg, James Whittaker and Karl Wiegers.

After 15 years in software development, I had the opportunity to focus on software testing for over 25 years. Specialized in testing process improvements, I founded and participated in the creation of multiple associations focused on software testing: AST (Association of Software Tester), ISTQB (International Software

1 Beizer, B. (1990). *Software Testing Techniques*, 2nd edition. ITP Media.

Testing Qualification Board), CFTL (Comité Français des Tests Logiciels, the French Software Testing committee) and GASQ (Global Association for Software Quality). I also dedicate these books to you, the reader, so that you can improve your testing competencies.

Preface

Implementation

In the first part of these two books on systems-of-systems testing, we identified the impacts of software development cycles, testing strategies and methodologies, and we saw the benefit of using a quality referential and the importance of test documentation and reporting. We have identified the impact of test levels and test techniques, whether we are talking about static techniques or dynamic techniques. We ended with an approach to test project management that allowed us to identify that human actor and how their interactions are essential elements that must be considered.

In this second part of the book on systems-of-systems testing, we will focus on more practical aspects such as managing test projects, testing processes and how to improve them continuously. We will see the additional but necessary processes such as the management of requirements, defects and configurations, and we will also see a case study allowing us to ask ourselves several useful questions. We will end with a perilous prediction exercise by listing the challenges that tests will have to face in the years to come.

August 2022

1

Test Project Management

We do not claim to replace the many contributions of illustrious authors on good practices in project management. Standards such as PMBOK (PMI 2017) or CMMI and methodologies such as ITIL and PRINCE2 comprehensively describe the tasks, best practices and other activities recommended to properly manage projects. We focus on certain points associated with the testing of software, components, products and systems within systems-of-systems projects.

At the risk of writing a tautology, the purpose of project management is to manage projects, that is, to define the tasks and actions necessary to achieve the objectives of these projects. The purpose, the ultimate objective of the project, takes precedence over any other aspect, even if the budgetary and time constraints are significant. To limit the risks associated with systems-of-systems, the quality of the deliverables is very important and therefore tests (verifications and validations that the object of the project has been achieved) are necessary.

Project management must ensure that development methodologies are correctly implemented (see Chapter 2) to avoid inconsistencies. Similarly, project management must provide all stakeholders with an image of the risks and the progress of the system-of-systems, its dependencies and the actions to be taken in the short and medium term, in order to anticipate the potential hazards.

1.1. General principles

Management of test projects, whether on components, products, systems or systems-of-systems, has a particularity that other projects do not have: they depend – for their deadlines, scope and level of quality – on other parts of the projects: the development phases. Requirements are often unstable, information arrives late, deadlines are shorter because they depend on evolving developments and longer

deadlines, the scope initially considered increases, the level of quality of input data – requirements, components to be tested, interfaces – is often of lower quality than expected and the number of faults or anomalies is greater than anticipated. All of these are under tighter budgetary and calendar constraints because, even if the developments take longer than expected, the production launch date is rarely postponed.

The methodologies offered by ITIL, PRINCE2, CMMI, etc. bring together a set of good practices that can be adapted – or not – to our system-of-systems project. CMMI, for example, does not have test-specific elements (only IVV), and it may be necessary to supplement CMMI with test-specific tasks and actions as offered by TMM and TMMI.

Let us see the elements specific to software testing projects.

1.1.1. *Quality of requirements*

Any development translates requirements (needs or business objectives) into a component, product or system that will implement them. In an Agile environment, requirements are defined in the form of User Stories, Features or Epics. The requirements can be described in so-called specification documents (e.g. General Specifications Document or Detailed Specifications Document). Requirements are primarily functional – they describe expected functionality – but can be technical or non-functional. We can classify the requirements according to the quality characteristics they cover as proposed in Chapter 5 of Volume 1 (Homès 2022a).

Requirements are provided to development teams as well as test teams. Production teams – design, development, etc. – use these requirements to develop components, products or systems and may propose or request adaptations of these requirements. Test teams use requirements to define, analyze and implement, or even automate, test cases and test scenarios to validate these requirements. These test teams must absolutely be informed – as soon as possible – of any change in the requirements to proceed with the modifications of the tests.

The requirements must be SMART, that is:

– **Specific**: the requirements must be clear, there must be no ambiguity and the requirements must be simple, consistent and with an appropriate level of detail.

– **Measurable**: it must be possible, when the component, product or system is designed, to verify that the requirement has been met. This is directly necessary for the design of tests and metrics to verify the extent to which requirements are met.

– **Achievable**: the requirements must be able to be physically demonstrated under given conditions. If the requirements are not achievable (e.g. the system will have 100% reliability and 100% availability), the result will be that the component, product or system will never be accepted or will be cost-prohibitive. Achievable includes that the requirement can be developed in a specific time frame.

– **Realistic**: in the context of software development – and testing – is it possible to achieve the requirement for the component, product or system, taking into account the constraints in which the project is developed? We add to this aspect the notion of time: are the requirements achievable in a realistic time?

– **Traceable**: requirements traceability is the ability to follow a requirement from its design to its specification, its realization and its implementation to its test, as well as in the other direction (from the test to the specification). This helps to understand why a requirement was specified and to ensure that each requirement has been correctly implemented.

1.1.2. *Completeness of deliveries*

The completeness of the software, components, products, equipment and systems delivered for the tests is obviously essential. If the elements delivered are incomplete, it will be necessary to come back to them to modify and complete them, which will increase the risk of introducing anomalies.

This aspect of completeness is ambiguous in incremental and iterative methodologies. On the one hand, it is recommended to deliver small increments, and on the other hand, losses should be eliminated. Small increments imply partial releases of functionality, thus generation of “losses” both regarding releases and testing (e.g. regression testing) – in fact, all the expectations related to these multiple releases and multiple test runs – to be performed on these components. Any evolution within the framework of an iteration will lead to a modification in the functionalities and therefore an evolution compared to the results executed during the previous iterations.

1.1.3. *Availability of test environments*

The execution of the tests is carried out in different test environments according to the test levels envisaged. It will therefore be necessary to ensure the availability of environments for each level.

The test environment is not limited to a machine on which the software component is executed. It also includes the settings necessary for the proper execution of the component, the test data and other applications – in the appropriate versions – with which the component interacts.

Test environments, as well as their data and the applications they interface with must be properly synchronized with each other. This implies an up-to-date definition of the versions of each system making up the system-of-systems and of the interfaces and messages exchanged between them.

Automating backups and restores of test environments allows testers to self-manage their environments so that they are not a burden on production systems management teams.

In DevOps environments, it is recommended to enable automatic creation of environments to test builds as they are created by developers. As proposed by Kim et al. (2016), it is necessary to allow to recreate – automatically – the test environments rather than trying to repair them. This automatic creation solution ensures an identical test environment to the previous version, which will facilitate regression testing.

1.1.4. *Availability of test data*

It is obvious that the input test data of a test case and the expected data at the output of a test case are necessary, and it is also important to have a set of other data that will be used for testing:

– data related to the users who will run the tests (e.g. authorization level, hierarchical level, organization to which they are attached, etc.);

– information related to the test data used (e.g. technical characteristics, composition, functionalities present, etc.) and which are grouped in legacy systems interfaced with the system-of-systems under test;

– historical information allowing us to make proposals based on this historical information (e.g. purchase suggestions based on previous purchases);

– information based on geographical positioning (e.g. GPS position), supply times and consumption volumes to anticipate stock replenishment needs (e.g. need to fill the fuel tank according to the way to drive and consume fuel, making it possible to offer – depending on the route and GPS information – one or more service stations nearby);

– etc.

The creation and provision of quality test data is necessary before any test campaign. Designing and updating this data, ensuring that it is consistent, is extremely important because it must – as far as possible – simulate the reality of the exchanges and information of each of the systems of the system-of-systems to be tested. We will therefore need to generate data from monitoring systems (from sensors, via IoT systems) and ensure that their production respects the expected constraints (e.g. every n seconds, in order to identify connection losses or deviations from nominal operating ranges).

Test data should be realistic and consistent over time. That is, they must either simulate a reference period and each of the campaigns must ensure that the systems have modified their reference date (e.g. use a fixed range of hours and reset systems at the beginning of this range) or be consistent with the time of execution of the test campaign. This last solution requires generating the test data during the execution of the test campaign, in order to verify the consistency of the data with respect to the expected (e.g. identification of duplicate messages, sequencing of messages, etc.) and therefore the proper functioning of the system-of-systems as a whole.

1.1.5. *Compliance of deliveries and schedules*

Development and construction projects are associated with often strict delivery dates and schedules. The impact of a late delivery of a component generates cascading effects impacting the delivery of the system and the system-of-systems. Timely delivery, with the expected features and the desired level of quality, is therefore very important. In some systems-of-systems, the completeness of the functionalities and their level of quality are often more important than the respect of the delivery date. In others, respecting the schedule is crucial in order to meet imperatives (e.g. launch window for a rocket aiming for another planet).

Test projects depend on the delivery of requirements and components to be tested within a specific schedule. Indeed, testers can only design tests based on the requirements, user stories and features delivered to them and can only run tests on the components, products and systems delivered to them in the appropriate test environments (i.e. including the necessary data and systems). The timely delivery of deliverables (contracts, requirements documents, specifications, features, user stories, etc.) and components, products and systems in a usable state – that is, with information or expected and working functionality – is crucial, or testers will not be able to perform their tasks properly.

This involves close collaboration between test manager and project managers in charge of the design and production of components, products or systems to be

tested, as well as managers in charge of test environments and the supply of test data.

In the context of Agile and Lean methods, any delay in deliveries and any non-compliance with schedules is a "loss of value" and should be eliminated. It is however important to note that the principles of agility propose that it is the development teams that define the scope of the functionalities to be delivered at each iteration.

1.1.6. *Coordinating and setting up environments*

Depending on the test levels, environments will include more and more components, products and systems that will need to coordinate to represent test environments representative of real life. Each environment includes one or more systems, components, products, as well as interfaces, ETLs and communication equipment (wired, wireless, satellite, optical networks, etc.) of increasing complexity. The design of these various environments quickly becomes a full-time job, especially since it is necessary to ensure that all the versions of all the software are correctly synchronized and that all the data, files, contents of databases and interfaces are synchronized and validated in order to allow the correct execution of the tests on this environment.

The activity of coordinating and setting up environments interacts strongly with all the other projects participating in the realization of the system-of-systems. Some test environments will only be able to simulate part of the target environment (e.g. simulation of space vacuum and sunlight with no ability to simulate zero gravity), and therefore there may be, for the same test level, several test execution campaigns, each on different technical or functional domains.

1.1.7. *Validation of prerequisites – Test Readiness Review (TRR)*

Testing activities can start effectively and efficiently as soon as all their prerequisites are present. Otherwise, the activities will have to stop and then start again when the missing prerequisite is provided, etc. This generates significant waste of time, not to mention everyone's frustration. Before starting any test task, we must make sure that all the prerequisites are present, or at the very least that they will arrive on time with the desired level of quality. Among the prerequisites, we have among others the requirements, the environment, the datasets, the component to be tested, the test cases with the expected data, as well as the testers, the tools and procedures for managing tests and anomalies, the KPIs and metrics allowing the reporting of the progress of the tests, etc.

One solution to ensure the presence of the prerequisites is to set up a TRR (Test Readiness Review) milestone, a review of the start of the tests. The purpose of this milestone is to verify – depending on the test level and the types of test – whether or not the prerequisites are present. If prerequisites are missing, it is up to the project managers to decide whether or not to launch the test activity, taking into account the identified risks.

In Agile methods, such a review can be informal and only apply to one user story at a time, with the acronym DOR for definition of ready.

1.1.8. *Delivery of datasets (TDS)*

The delivery of test datasets (TDS) is not limited to the provision of files or databases with information usable by the component, product or system. This also includes – for the applications, components, products or systems with which the component, product or system under test interacts – a check of the consistency and synchronization of the data with each other. It will be necessary to ensure that the interfaces are correctly described, defined and implemented.

Backup of datasets or automation of dataset generation processes may be necessary to allow testers to generate the data they need themselves.

The design of coherent and complete datasets is a difficult task requiring a good knowledge of the entire information system and the interfaces between the component, product or system under test on the one hand and all the other systems of the test environment on the other hand. Some components, products or systems may be missing and replaced by “stubs” that will simulate the missing elements. In this case, it is necessary to manage these “stubs” with the same rigor as if they were real components (e.g. evolution of versions, data, etc.).

1.1.9. *Go-NoGo decision – Test Review Board (TRB)*

A Go-NoGo meeting is used to analyze the risks associated with moving to the next step in a process of designing and deploying a component, product, system or system-of-systems, and to decide whether to proceed to the next step.

This meeting is sometimes split into two reviews in time:

– A TRB (Test Review Board) meeting analyzes the results of the tests carried out in the level and determines the actions according to these results. This technical meeting ensures that the planned objectives have been achieved for the level.

– A management review to obtain – from the hierarchy, the other stakeholders, the MOA and the customers – a decision (the "Go" or the "NoGo" decision) accepted by all, with consideration of business risks, marketing, etc.

The Go-NoGo meeting includes representatives from all business stakeholders, such as operations managers, deployment teams, production teams and marketing teams.

In an Agile environment, the concept of Go-NoGo and TRB is detailed under the concept of DOD (definition of done) for each of the design actions.

1.1.10. *Continuous delivery and deployment*

The concept of continuous integration and continuous delivery (CI/CD) is interesting and deserves to be considered in systems-of-systems with preponderant software. However, such concepts have particular constraints that we must study, beyond the use of an Agile design methodology.

1.1.10.1. *Continuous delivery*

The continuous delivery practices mentioned in Kim et al. (2016) focus primarily on the aspects of continuous delivery and deployment of software that depend on automated testing performed to ensure developers have quick (immediate) feedback on the defects, performance, security and usability concerns of the components put in configuration. In addition, the principle is to have a limited number of configuration branches.

In the context of systems-of-systems, where hardware components and subsystems including software must be physically installed – and tested on physical test benches – the ability to deliver daily and ensure the absence of regressions becomes more complex, if not impossible, to implement. This is all the more true since the systems-of-systems are not produced in large quantities and the interactions are complex.

1.1.10.2. *Continuous testing*

On-demand execution of tests as part of continuous delivery is possible for unit testing and static testing of code. Software integration testing could be considered, but anything involving end-to-end (E2E) testing becomes more problematic because installing the software on the hardware component should generate a change in the configuration reference of the hardware component.

Among the elements to consider, we have an ambiguity of terminology: the term ATDD (Acceptance Test-Driven Development) relates to the acceptance of the software component alone, not its integration, nor the acceptance of the system-of-system nor of the subsystem or equipment.

Another aspect to consider is the need for test automation and (1) the continued increase in the number of tests to be executed, which will mean increasing test execution time as well as (2) the need to ensure that the test classes in the software (case of TDD and BDD) are correctly removed from the versions used in integration tests and in system tests.

One of the temptations associated with testing in a CI/CD or DevOps environment is to pool the tests of the various software components into a single test batch for the release, instead of processing the tests separately for each component. This solution makes it possible to pool the regression tests of software components, but is a difficult practical problem for the qualification of systems-of-systems as mentioned in Sacquet and Rochefolle (2016).

1.1.10.3. *Continuous deployment*

Continuous deployment depends on continuous delivery and therefore automated validation of tests, and the presence of complete documentation – for component usage and administration – as well as the ability to run end-to-end on an environment representative of production.

According to Kim et al. (2016), in companies like Amazon and Google, the majority of teams practice continuous delivery and some practice continuous deployment. There is wide variation in how to perform continuous deployment.

1.2. Tracking test projects

Monitoring test projects requires monitoring the progress of each of the test activities for each of the systems of the system-of-systems, as well as on each of the test environments of each of the test levels of each of these systems. It is therefore important that the progress information of each test level is aggregated and summarized for each system and that the test progress information of each system is aggregated at the system-of-systems level. This involves defining the elements that must be measured (the progress), against which benchmark they must be measured (the reference) and identifying the impacts (dependencies) that this can generate. Reporting of similar indicators from each of the systems will facilitate understanding. Automated information feedback will facilitate information retrieval.

1.3. Risks and systems-of-systems

Systems-of-systems projects are subject to more risk than other systems in that they may inherit upstream-level risks and a process's tolerance for risk may vary by organization and the delivered product. In Figure 1.1, we can identify that the more we advance in the design and production of components by the various organizations, the risks will be added and the impact for organizations with a low risk tolerance will be more strongly impacted than others.

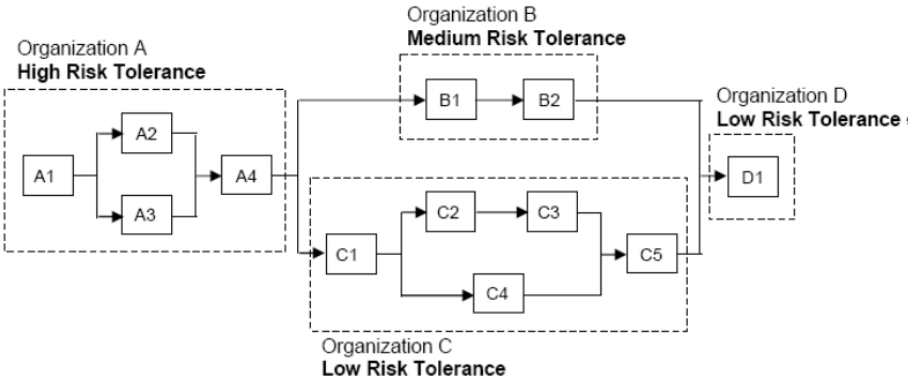

Figure 1.1. *Different risk tolerance*

In Figure 1.2, we can identify that an organization will be impacted by all the risks it can inherit from upstream organizations and that it will impose risks on all downstream organizations.

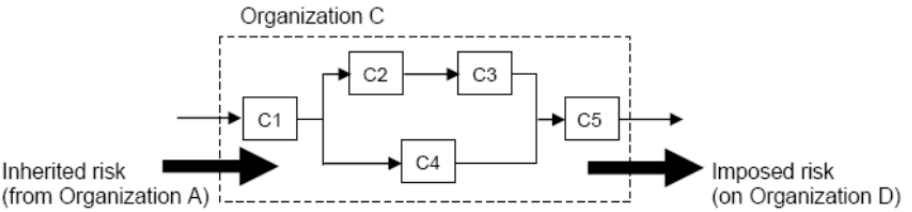

Figure 1.2. *Inherited and imposed risks*

We realize that risk management in systems-of-systems is significantly more complex than in the case of complex systems and may need to be managed at multiple levels (e.g. interactions between teams, between managers of the project or between the managers – or leaders – of the organizations).

1.4. Particularities related to SoS

According to Firesmith (2014), several pitfalls should be avoided in the context of systems-of-systems, including:

– inadequate system-of-systems test planning;

– unclear responsibilities, including liability limits;

– inadequate resources dedicated to system-of-systems testing;

– lack of clear systems-of-systems planning;

– insufficient or inadequate systems-of-systems requirements;

– inadequate support of individual systems and projects;

– inadequate cross-project defect management.

To this we can add:

– different quality requirements according to the participants/co-contractors, including regarding the interpretation of regulatory obligations;

– the needs to take into account long-term evolutions;

– the multiplicity of level versions (groupings of software working and delivered together), multiple versions and environments;

– the fact that systems-of-systems are often unique developments.

1.5. Particularities related to SoS methodologies

Development methodologies generate different constraints and opportunities. Sequential developments have demonstrated their effectiveness, but involve constraints of rigidity and lack of responsiveness, if the contexts change. Agility offers better responsiveness at the expense of a more restricted analysis phase and an organization that does not guarantee that all the requirements will be developed. The choice of a development methodology will imply adaptations during the management of the project and during the testing of the components of the system-of-systems.

Iterative methodologies involve rapid delivery of components or parts of components, followed by refinement phases if necessary. That implies that:

– The planned functionalities are not fully provided before the last delivery of the component. Validation by the business may be delayed until the final delivery of the component. This reduces the time for detecting and correcting anomalies and

can impact the final delivery of the component, product or system, or even the system-of-systems.

– Side effects may appear on other components, so it will be necessary to retest all components each time a component update is delivered. This solution can be limited to the components interacting directly with the modified component(s) or extend to the entire system-of-systems, and it is recommended to automate it.

– The interfaces between components may not be developed simultaneously and therefore that the tests of these interfaces may be delayed.

Sequential methodologies (e.g. V-cycle, Spiral, etc.) focus on a single delivery, so any evolution – or need for clarification – of the requirement will have an impact on lead time and workload, both in terms of development (redevelopment or adaptation of components, products or systems) and in terms of testing (design and execution of tests).

1.5.1. *Components definition*

Within the framework of sequential methodologies, the principle is to define the components and deliver them finished and validated at the end of their design phase. This involves a complete definition of each product or system component and the interactions it has with other components, products or systems. These exhaustive definitions will be used both for the design of the component, product or system and for the design of the tests that will validate them.

1.5.2. *Testing and quality assurance activities*

It is not possible to envisage retesting all the combinations of data and actions of the components of a level of a system-of-systems; this would generate a workload disproportionate to the expected benefits. One solution is to verify that the design and test processes have been correctly carried out, that the proofs of execution are available and that the test activities – static and dynamic – have correctly covered the objectives. These verification activities are the responsibility of the quality assurance teams and are mainly based on available evidence (paper documentation, execution logs, anomaly dashboards, etc.).

1.6. Particularities related to teams

In a test project, whether it is software testing or systems-of-systems testing, one element to take into account is the management of team members, and their