COMPUTER ENGINEERING SERIES

Advanced Testing of Systems-of-Systems 1

Theoretical Aspects

Bernard Homès

iSTE

WILEY

Advanced Testing of Systems-of-Systems 1

Advanced Testing of Systems-of-Systems 1

Theoretical 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: 2022943899

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-78630-749-1

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

The breadth of the subject justifies splitting this work in two books. Part I, this book, covers the general aspects applicable to systems-of-systems testing, among them the impact of development life cycle, test strategy and methodology, the added value of quality referential, test documentation and reporting. We identified the impact of various test levels and test techniques, whether static or dynamic.

In the second book, we will focus on project management, identifying human interactions as primary elements to consider, and we will continue with practical aspects such as testing processes and their iterative and continuous improvement. We will also see additional but necessary processes, such as requirement management, defects management and configuration management. In a case study, we will be able to ask ourselves several useful questions. We will finish this second book with a rather perilous prospective exercise by listing the challenges that testing will need to face in the coming years.

These two books make a single coherent and complete work building on more than 40 years of experience by the author. The main aspect put forward is the difference between the traditional vision of software testing – focused on one system and one version – and the necessary vision when multiple systems and multiple versions of software must be interconnected to provide a service that needs to be tested thoroughly.

August 2022

1

Introduction

1.1. Definition

There are many definitions of what a system-of-systems (or SoS) is. We will use the following one: "A system-of-systems is a set of systems, software and/or hardware, developed to provide a service by collaborating together, by organizations that are not under the same management". This simple definition entails challenges and adaptations that we will identify and study.

A system-of-systems can be considered from two points of view: on the one hand, from the global systemic level (we could take the image of a company information system) and, on the other hand, from the unitary application system (which we may call a subsystem, application system or application, software-predominant equipment or component). We will thus have at the upper level a system-of-systems that could be a "information system" that is made of multiple systems that we will call subsystems. For example, a company may have in their information system one accounting system, a CRM, a human resource management system, a stock management system, etc. These different systems are most likely developed by different editors and their interaction provides a service to the company. Other examples of systems-of-systems are air traffic systems, aircrafts and satellite systems, vehicles and crafts. In these systems-of-systems, the service is provided to the users when all subsystems work, correctly and quickly exchanging data between them.

Systems-of-systems, even if they are often complex, are intrinsically different from complex systems: a complex system, such as an operating system, may be developed by a single organization (see Figure 1.1) and thus does not respond exactly to the definition as the subsystems are developed under the same hierarchy. The issue of diverse organizations and directions (see Figure 1.2) implies technical, economic and financial objectives that may diverge between the parties and thus multiple separate systems creating, when put together, a system-of-systems. A more exhaustive description is presented in ISO21840 (2019).

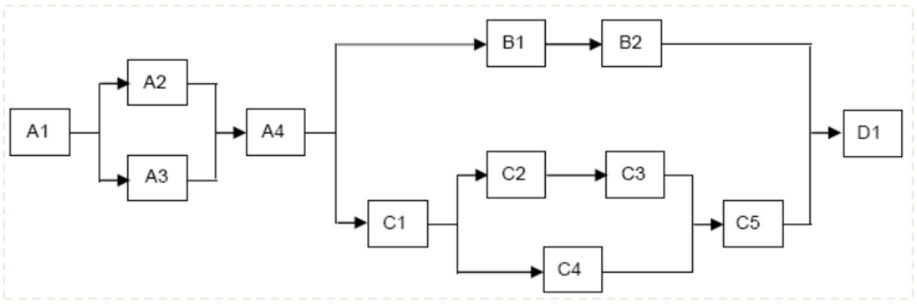

Figure 1.1. *Complex system*

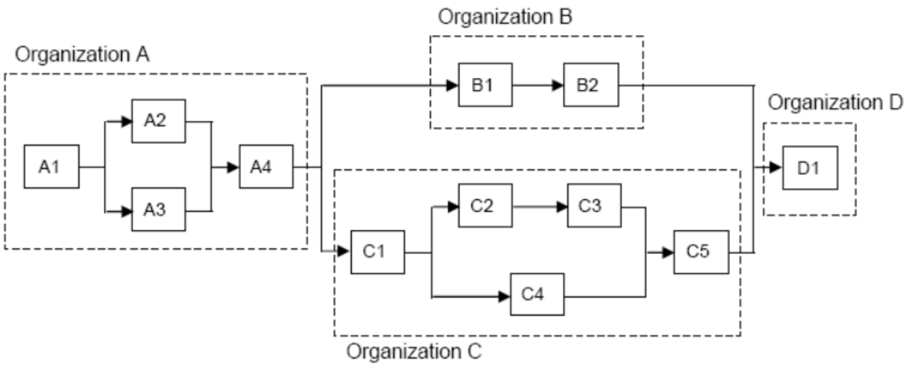

Figure 1.2. *System-of-systems*

Usually, a system-of-systems tend to have:

– multiple levels of stakeholders, sometimes with competing interests;

– multiple and possibly contradictory objectives and purposes;

– disparate management structures whose limits of responsibility are not always clearly defined;

– multiple life cycles with elements implemented asynchronously, resulting in the need to manage obsolescence of subsystems;

– multiple owners – depending on subsystems – making individual resource and priority decisions.

It is important to note that the characteristics differ between systems and systems-of-systems and are not mutually exclusive.

1.2. Why and for who are these books?

1.2.1. *Why?*

Why a book on the testing of systems-of-systems? Systems-of-systems are part of our everyday life, but they are not addressed in software testing books that focus only on one software at a time, without taking into account the physical systems that are required to execute them, nor the interactions between them that increase the difficulty and combinatorial complexity of testing. To ensure quality for a system-of-systems means to ensure for each subsystem (and sub-subsystem) the quality of the design process for each of these systems, subsystems, components, software, etc., that make them up.

Frequently, actors on a system-of-systems project focus only on their own activity, resecting contractual obligations, without considering the requirements of the overall system-of-systems or the impact their system may have on the system-of-systems. This focus also applies when developing software to be used in a company's information system: the development teams seldom exchange with the teams in charge of support or production. This slowly changes with the introduction of DevOps in some environments, but the gap between IT and business domains remains large.

As more projects become increasingly complex, connected to one another in integrated systems-of-systems, books on advanced level software testing in the frame of these kinds of systems become necessary.

Most books on software testing focus on testing one software for one structure, where those that define requirements, design the software and test it are in the same organization, or – at least – under the same hierarchy. These are thus a common point for decisions. In a system-of-systems, there are at least two sets of organizations: the client and the contractors. A contractual relationship exists and directs the exchanges between these organizations.

Many specific challenges are associated with these contractual relationships:

– Are requirements and specifications correctly defined and understood by all parties?

– Are functionalities and technical characteristics coherent with the rest of the system-of-systems with which the system will be merged?

– Have evolutions, replacements and possible obsolescence been considered for the whole duration of the system-of-systems being developed?

In a system-of-systems, interactions with other systems are more numerous than in a simple system. Thus, the verification of these numerous exchanges between components and systems will be a heavier load than for other software. In case of defect, it will be necessary to identify which party will have to implement the fixes, and each actor will prefer to reject the responsibility to others. These decisions may be influenced by economic factors (it may be cheaper to fix one system instead of another), regulatory factors (conformance may be easier to demonstrate on one system instead of another), contractual or technical (one system may be simpler to change than another).

Responsibilities are different between the client and the organization that executes the development. The impact is primarily felt by the client, and it is up to the development organization to ensure the quality of the developments.

The increase in the complexity of IT solutions forces us to envisage a more efficient management of specific challenges linked to systems-of-systems to which we are increasingly dependent.

1.2.2. *Who is this book for?*

Design of software, systems and systems-of-systems requires interaction between many individuals, each with different objectives and different points of view. The notion of "quality" of a deliverable will vary and depend on the relative position of each party. This book tries to cover each point of view and shows the major differences between what is described in many other books – design and test of a single software application – with regard to the complexity and reality of systems-of-systems. The persons who could benefit from reading this book are as follows:

– design organization project managers who must ensure that the needs of users, their customers and their clients are met and therefore that the applications, systems and systems-of-systems are correctly developed and tested (i.e. verified and validated);

– by extension, the design organization we will have assistant Project Managers, who will have to ensure that the overall objectives of the designing organization are correctly checked and validated, especially taking into account the needs of the users – forever changing given the length of systems-of-systems projects – and that the evidence provided to justify a level of quality is real;

– customer project managers, whether for physical (hardware) production or for digital (software) production, and specifically those responsible for programs, development projects or test projects, in order to ensure that the objectives of Design organizations are correctly understood, deduced and implemented in the solutions they put in place;

– test managers in charge of quality and system-of-systems testing (at design organization level), as well as test managers in charge of quality and system testing (at design and at client level), applications and predominant software components entering into the composition of systems-of-systems, with the particularity that the so-called "end-to-end" (E2E) tests are not limited to a single application or system, but cover all the systems making up the system-of-systems;

– testers, test analysts and technical test analysts wishing to obtain a more global and general vision of their activities, to understand how to implement their skills and knowledge to further develop their careers;

– anyone wishing to develop their knowledge of testing and their impact on the quality of complex systems and systems-of-systems.

1.2.3. *Organization of this book*

These books are part of a cycle of three books on software testing:

– the first book (*Fundamentals of Software Testing*, ISTE and Wiley, 2012) focuses on the ISTQB Foundation level tester certification and is an aid to obtaining this certification; it was elected third best software testing book of all time by BookAuthority.org;

– this present book on the general aspects of systems-of-systems testing;

– a third book on practical implementation and case studies showing how to implement tests in a system-of-systems, *Advanced Testing of Systems-of-Systems 2: Practical Aspects* (ISTE and Wiley, 2022).

The last two books complement each other and form one. They are independent of the first.

1.3. Examples

We are in contact with and use systems-of-systems of all sizes every day: a car, an orchestra, a control-command system, a satellite telecommunications system, an air traffic control management system, an integrated defense system, a multimodal transport system, a company, all are examples of systems-of-systems. There is no single organizational hierarchy that oversees the development of all the components integrated into these systems-of-systems; some components can be replaced by others from alternative sources.

In this book, we will focus primarily on software-intensive systems. We use them every day: a company uses many applications (payroll, inventory management, accounting, etc.) developed by different companies, but which must work together. This company information system is thus a system-of-systems.

Our means of transportation are also systems-of-systems: the manufacturers (of metros, cars, planes, trains, etc.) are mainly assemblers integrating hardware and software designed by others.

Operating systems – for example, open source – integrating components from various sources are also systems-of-systems. The developments are not carried out under the authority of an organization, and there is frequently integration of components developed by other structures.

The common elements of systems-of-systems – mainly software-intensive systems – are the provision of a service, under defined conditions of use, with expected performance, providing a measurable quality of service. It is important to think "systems" at the level of all processes, from design to delivery to the customer(s) of the finished and operational system-of-systems.

Often, systems-of-systems include, within the same organization, software of various origins. For example, CRM software such as SAP, a Big-Data type data analysis system, vehicle fleet management systems, accounting monitoring or analysis of various origins, load sharing systems (load balancing), etc.

The examples in this book come from the experience of the author during his career. We will therefore have examples in space, military or civil aeronautics, banking systems, insurance and manufacturing.

To fully understand what a system-of-system is in our everyday life, let's take the example of connecting your mobile phone to your vehicle. First of all, we have your vehicle, and the operating system which interacts via a Bluetooth connection with your phone. Then, we have your phone, which has an operating system version that evolves separately from your car; then, we have the version of the software app which provides the services to your phone and is available on a store. Finally, we have the subscription that your car manufacturer provides you with to ensure the connection between your vehicle and your phone. This subscription is certainly supported by a series of mainframes, legacy applications and these must also be accessible via the Web. The information reported by your vehicle will certainly be included in a repository (Big Data, Datalake, etc.) where it can be aggregated and allow maintenance of your vehicle as well as improvement in the maintenance of vehicles of your type. This maintenance information will allow your dealer to warn you if necessary (e.g. failure identified while the vehicle is not at the garage, and need to go to a garage quickly). You can easily identify all the systems that need to communicate correctly so that you – the user – are satisfied with the solution offered (vehicle + mobile + application + subscription + information reported + emergency assistance + vehicle monitoring + preventive or corrective maintenance + … etc.).

1.4. Limitations

This book will focus primarily on systems-of-systems and software-intensive systems, and how to test such systems. The identified elements can be extrapolated to physical systems-of-systems.

As we will focus on testing, the view we will have of systems-of-systems will be that of Test Managers: either the person in charge of testing for the client or for the design organization, or in charge of testing a component, product, or subsystem of a system-of-systems, in order to identify the information to be provided within the framework of a system-of-systems. We will also use this view of the quality of systems and systems-of-systems to propose improvements to the teams in charge of implementation (e.g. software development teams, developers, etc.).

This work is not limited to the aspects of testing – verification and validation – of software systems, but also includes the point of view of those in charge of improving the quality of components – software or hardware – and processes (design, maintenance, continuous improvement, etc.).

As part of this book, we will also discuss the delivery aspects of systems-of-systems in the context of DevOps.

1.5. Why test?

The necessity of testing the design of software, components, products or systems before using or marketing them is evident, known and recognized as useful. The objective of the test can be seen according to a system of five successive phases, as proposed by Beizer (1990):

– testing and debugging are related activities in that it is necessary to test in order to be able to debug;

– the purpose of the test is to show the proper functioning of the software, component, product or system;

– the purpose of the test is to show that the software, component, product or system does not work;

– the objective of the test is not to prove anything, but to reduce the perceived risk of non-operation to an acceptable value;

– the test is not an action; it is a mental discipline resulting in software, components, products or systems having little risk, without too much testing effort.

Each of these five phases represents an evolution of the previous phases and should be integrated by all stakeholders on the project. Any difference in the understanding of "Why we test" will lead to tensions on the strategic choices (e.g. level of investment, prioritization of anomalies and their criticalities, level of urgency, etc.) associated with testing.

A sixth answer to the question "why test?" adds a dimension of improving software quality and testing processes to identify anomalies in products comprising software such as systems-of-systems. This involves analyzing the causes of each failure and implementing processes and procedures to ensure the non-reproducibility of this type of failure. In critical safety areas (e.g. aeronautics), components are added to the systems to keep information on the operating status of the systems in the event of a crash (the famous "black boxes"). The analysis of these components is systematic and makes it possible to propose improvements in procedures or aircraft design, so as to make air travel even more reliable.

Adding such a way of doing things to development methods is what is planned during sprint retrospectives (Agile Scrum methodology) and more generally in feedback activities. This involves objectively studying anomalies or failures and improving processes to ensure that they cannot recur.

1.6. MOA and MOE

When talking about systems-of-systems, it is common (in France) to use the terms client project management (MOA) and designer project management (MOE). These acronyms from cathedral building have been taken up in the world of software engineering. They are 100% French-speaking, and represent two different views of the same things:

– the client project owner (abbreviated MOA) represents the end users that have the need and define the objectives, schedule and budget; MOA is responsible for the needs of the company, of the users and of their customers, of the principals, sponsors or stakeholders, of the business of the company. There usually is only one MOA;

– the designer project manager (abbreviated MOE) represents the person (or company) who designs and controls the production of an element or a set of elements making up the system-of-systems; it is all the production teams, with constraints and objectives often different from those of the company and the principals. There could be multiple MOE.

In a system-of-systems, we therefore must take into account this separation between MOA (client) and MOE (supplier) and therefore the two separate views of each of these major players.

When we deal with systems-of-systems testing, we will speak of "test manager", but these can be assigned to a single test level (e.g. for a software subsystem) or cover several levels (e.g. the manager responsible for testing at the project management level).

1.7. Major challenges

Recent statistics[1] show that only 6% of large IT projects are successful and 52% are outside budget, timeframe or lack all the expected functionalities. The remaining 42% are cancelled before their delivery, becoming losses for the organizations.

We can conclude that the most appropriate development and testing processes should be implemented to minimize, as much as possible, the risks associated with systems-of-systems. When compared to complex systems, Test Managers of systems-of-systems face and must master many challenges.

1.7.1. *Increased complexity*

Systems-of-systems are generally more complex and larger than complex systems developed by a single entity. We must consider:

– interfaces and interoperability of systems with each other, both logical (messages exchanged, formats, coding, etc.) and physical (connectors, protections against EMP, length of connectors, etc.);

– development life cycles of the organizations and their evolutions;

– obsolescence of components of the system-of-systems, as well as their versions and compatibilities;

1 According to https://www.standishgroup.com/sample_research_files/BigBangBoom.pdf.

– integration of simulation and decision support tools, as well as the representativity of these tools with regard to the components they simulate;

– governance and applicable standards – as well as their implementation – for both process and product aspects;

– design architecture and development process frameworks;

– the quality of requirements and specifications, as well as their stability or evolution over time;

– the duration of the design process to develop and integrate all the components, compatibility of these with each other, as well as their level of security and the overall security of the entire system-of-systems;

– organizational complexity resulting from the integration of various organizations (e.g. following takeovers or mergers) or the decision to split the organizations, to call on relocated external subcontracting (offshore) or not;

– the complexity of development cycles stemming from the desire to change the development model, which implies the coexistence of more or less incompatible models with each other for fairly long periods.

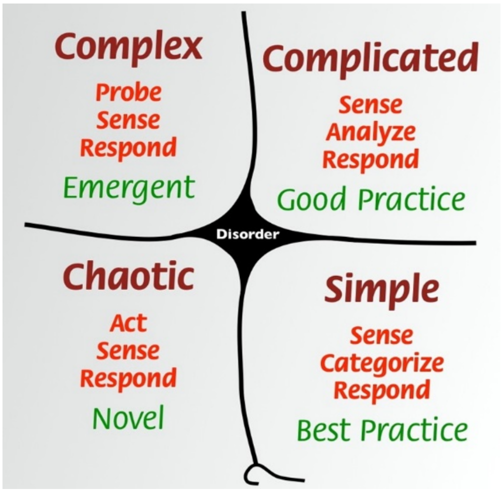

Figure 1.3. *Simple–complicated–complex–chaotic*

We could use the CYNEFIN[2] model (see Figure 1.3, simple–complicated–complex–chaotic) to better understand the aspects of evolution between simple

2 https://www.le-blog-des-leaders.com/cynefin-framework/.

systems (most software developments), complicated systems (e.g. IT systems), complex systems (the majority of systems-of-systems) and chaotic systems, where the number of interactions is such that it is difficult (impossible?) to reproduce and/or simulate all the conditions of execution and operation of the system-of-systems.

To determine if the system is simple, complicated, complex or chaotic, we can focus on the predictability of effects and impacts. We also have the "disorder" state which is the initial position from which we will have to ask ourselves questions to determine which model of system we should turn to.

1.7.1.1. *Simple*

If the causes and effects are well known and predictable, the problem is said to be "simple". The steps can be broken down into feeling, categorizing and then acting. We can look at the applicable "best practices" and select the one(s) that is(are) appropriate, without needing to think too much.

1.7.1.2. *Complicated*

An environment will be said to be "complicated" when the causes and effects are understandable but require a certain expertise to understand them. The domain of practices – including software testing practices – is that of "best practices", known to experts and consultants and making it possible to reach a predefined final target.

1.7.1.3. *Complex*

In the realm of the "complex", the causes and effects are difficult to identify, to understand, to isolate and to define. It seems difficult, if not impossible, to get around the question. We are moving here from the field of "best practices" to that of the emergence of solutions appearing little by little, without an a priori identification of the final target. We are no longer here in a posture of expertise but are entering into a posture of a coach who asks questions, who enlightens through reflections and makes the actors gain understanding.

1.7.1.4. *Chaotic*

In a so-called "chaotic" system, we are unable to distinguish the links between causes and their effects. At this level, the reaction will often be an absence of reaction, like paralysis. When you're in chaos, the only thing you can do is get out of the chaos as quickly as possible, by any means imaginable. Given the exceptional side of what is happening, there are no best practices to apply. You will also not have the time to consult experts who will take a few weeks to analyse in detail what is happening and finally advise you on the right course of action. You will certainly not have the time to do a few harmless experiments to let an original solution emerge. The urgency is to take shelter: the urgency is to act first.

1.7.2. *Significant failure rate*

Most systems-of-systems are large – even very large – projects. Measured in function points (e.g. IFPUG or SNAP), these projects easily exceed 10,000 function points and even reach 100,000 function points. Capers Jones (2018a) tells us that on average these projects have a 31–47% probability of failure. The Chaos Report in 2020 confirms this trend with 19% of projects failing and 50% seriously off budget, off deadline or lacking in quality.

Since the causes of failure add up to one another, it is critical to implement multiple quality improvement techniques throughout the project, from the start of the project. The choice of these techniques should be made based on their measured and demonstrated effectiveness (i.e. not according to the statements or opinions of one or more individual). A principle applicable to QA and testing is “prevention is better than cure”. It is better to detect a defect early and avoid introducing it into any deliverable (requirements, codes, test cases, etc.), rather than discovering it late. This principle also applies to tests: reviews and inspections have demonstrated their effectiveness in avoiding the introduction of defects (measured effectiveness greater than 50%), while test suites generally only have an effectiveness of less than 35%. This is the basis of the “shift left” concept which encourages finding defects as early as possible (to the left in the task schedule). This justifies providing stakeholders with information on the level of quality of systems-of-systems from the start of design, as well as measurable information for each of the subsystems that compose them. Implementing metrics and a systematic reporting of measures is therefore necessary to prevent dangerous drifts from appearing and leading the project to failure.

1.7.3. *Limited visibility*

Since systems-of-systems are large projects involving several organizations, it is difficult to have complete and detailed visibility into all the components and their interactions with each other. It will be necessary to use documentation – paper or electronic via tools – to transmit the information. In this type of development, these activities will sometimes be taken over by those in charge of Quality Assurance. Test Managers belong to Quality Assurance, focusing mainly on the execution of tests to verify and validate requirements and needs.