

Scenarios in System Development

Ian Alexander

Scenarios are a powerful antidote to the complexity of systems and analysis. Telling stories about systems helps ensure that people—stakeholders—share a sufficiently wide view to avoid missing vital aspects of problems. Scenarios vary from brief stories to richly structured analyses, but are almost always based on the idea of a sequence of actions carried out by intelligent agents. People are very good at reasoning from even quite terse stories, for example detecting inconsistencies, omissions, and threats with little effort. These innate human capabilities give scenarios their power. Scenarios are applicable to systems of all types, and may be used at any stage of the development life cycle for different purposes.

Scenarios are simple, human things. There are many possible variations on the theme. The basic idea is just a story: someone does this, someone else does that:

The driver walks towards the car and presses his key. The car recognises the driver, unlocks the doors, and adjusts the driving seat, steering wheel, radio, and mirrors to the driver's preferred settings.

Scenarios are arguably the starting point for all modelling and design. Since systems either do something that somebody wants, or are shelf-ware, and scenarios describe how to do things, it seems hard to disagree with the idea that scenarios are the way to begin a development.

More and more modelling notations are being invented—I saw one in a new gadget being shown off at a trade fair, and it had no fewer than 26 symbols that somebody believed were necessary for requirements analysis—and no end to the madness is in sight. I have no idea how the developers of such things imagine that ordinary people are going to use anything so intricate, and so far removed from life. I do have a lively idea of the blank looks I'd get from practical engineers if I tried to pull any such trick on them.

The use of the narrative scenario in engineering seems in one way to be a kind of reaction against too much technology, too fast. There is no need to be a Luddite to wonder what is being missed in the race to construct ever more complex, formal, and unfamiliar models for ever more risky projects.

Scenarios allow us to take a backward glance. They use a simple, traditional activity—storytelling—to provide a vital missing element, namely a view of the whole of a situation. And they do this at low cost, at least once people are trained in their use.

The scenario perspective looks at what people do and how people use systems in their work, with concrete instead of abstract descriptions, focus on particular instances rather than generic types, being work- and not technology-driven, open-ended and fragmentary rather than complete and exhaustive, informal, rough, and colloquial instead of formal and rigorous, and with envisioned rather than specified outcomes.

Analysis means 'dissolving [into component particles]', which is fine and very necessary; but it also means looking at details, which as everyone knows is a way of not seeing the wood for the trees. Engineers love analysis and design: our profession's occupational hazard is diving into detail, ignoring the people involved, and what they may want.

Using scenarios in analysis is thus paradoxical. Analysis is about refinement, precision, and completeness with respect to the parts of a problem. But scenarios are basically holistic. Whether in terse and summary form, or written out at length in a carefully studied sequence—or even in a complex analytical framework with multiple paths ingeniously fitted together—the scenario is in essence, a single thing that conveys a human meaning. And that meaning is enhanced by the reader from her own experience; the story told in the written scenario slips effortlessly into the context of the network of meaning in the reader's mind.

"What are you doing?" sobbed the Djinni.

"I'm throwing you back into the sea", said the Fisherman. "Let me out of this bottle" wailed the desperate Djinni, "and I'll make you richer than King Solomon".

"You are a tricky Djinni", answered the Fisherman. "You deserve the same fate as the King in the story of The King and the Doctor."

"What story is that?" inquired the Djinni.

Stories are quite insistent on one point: a tale is not over until it's finished in every detail. The Djinni is not just playing for time by exploring side issues: he's thinking out other options and tricks that might result in a better outcome—from his dark and devious point of view. It goes without saying that the exploration is by storytelling, and the Fisherman has to use all his cunning to outwit his immensely powerful opponent. Scenario-based techniques such as searching for Exceptions, Functional Hazard Analysis, and Misuse Cases make use of the power of storytelling to explore likely weaknesses, and thus to make systems more reliable, safer, and more secure in the face of active and intelligent opposition.

Humanistic knowledge is unlike instrumental knowledge, since it aims to fashion a comprehensible narrative. The Internet and electronic media detract from the ability to concentrate in the way that a printed book encourages; the private world that a good book lets you into is a doorway into civilisation, as it fosters just that depth of thought that the instant availability of colourful hypertext banishes. Narrative is extremely important.

We believe that at least one kind of instrumental knowledge, that needed to serve system designers, testers, safety engineers, and others in their work by giving them a mental picture of how the system-to-be is going to be used, also needs to be fashioned into a 'comprehensible narrative'. Our goal is not, indeed, anything as grand as enabling humanity to grow emotionally or spiritually, as literature may do; but more humbly, we hope that scenarios may help engineers to make systems that serve humanity a little better.

Narrative can fulfil these diverse purposes because sequences of events in time are at the heart of our ability to construct meaning.

In David Attenborough's wonderful television series, *The Life of Mammals*, one story shows a group of monkeys of different species feeding on the ground. Suddenly one spots a snake. At once, all the members of the group shout out the 'Snake' call of their own species. Everyone looks about for the snake. The youngest realise the danger, learn the calls, and live to watch out for snakes another day.

The story:

"A snake slithers silently up to our group and grabs one of us."

could hardly be more elemental. The variation:

"but someone sees it, gives the 'Snake' alarm call, and we all escape."

could hardly carry a clearer survival message. And this tale of communication in the face of danger is part of our shared primate heritage. Actually there is a further twist: some monkeys give not only the basic call, but accompany it with a modifier, which seems to mean something like 'serious', 'intense'—the monkeys have not only nouns but adjectives to convey their meaning.

Human language is a far richer medium for sharing meanings than even the most sophisticated calls of our primate relatives, but Stephen Pinker is surely right when he writes in *The Language Instinct* that it grew out of the need to share knowledge with similar survival value. He quotes a conversation, imagined by the evolutionary biologist George Williams, between a mother and a child along the lines of,

"Don't play with that, dear, that's a Sabre-Toothed Tiger and it wants to eat you".

The rest of the scenario hardly needs to be told; the child that understood grew up safe and well, and the one that didn't, didn't.

Every parent of young children has made the same striking observation as I have: even very little ones instantly appreciate being shown different kinds of beast, and never tire of hearing about the noises they make, or of seeing how the crocodile snaps his jaws. These things are important to know if you're inexperienced and want to live.

It's no exaggeration, then, to suggest that spoken language, and with it the human mind, evolved precisely because planning—constructing and evaluating scenarios—is a survival skill.

Planning becomes important when resources are short, when outcomes are uncertain, and especially when intelligent opponents are making their own plans that contradict our own. That's pretty much all the time. We are social animals; gangs and tribes and shifting alliances and treachery are a permanent feature of our existence. Our social brain—the one that enjoys watching the scheming complexities of Shakespearean tragedies and soap operas—is adapted precisely to understand stories. It is adept at filling in details from

scant evidence; at guessing and reasoning about people's intentions; at predicting how people would respond if we chose a certain course of action.

The scenario exploits exactly these mental abilities. It uses narrative—a time-threaded sequence of actions. It focuses on agents and their (inter)actions. It is at root brief and abstract, but highly suggestive of context. It is helpful for predicting outcomes. It is all about courses of action.

In a nutshell, scenario approaches allow projects to benefit from the inherent strengths of narrative, and from the excellent match of the story form to the story-processing capabilities of the human brain—whether trained or not. We predict that scenarios will always be useful in engineering.

Scope—What Does Scenario Mean, and What Does it Cover?

Like other important concept terms in systems engineering, Thee word “Scenario” suggests a variety of different things, all subtly different. Fortunately, these form quite a coherent set of meanings, which we'll explore below.

One other concept intimately connected with scenarios is that of the Stakeholder. Scenarios are all about communicating with people. One of the main reasons why traditional development approaches failed was precisely that they excluded non-engineers. Quite often the kinds of models used were not too helpful for communicating with other engineers either. So we'll start by defining what we mean by Stakeholder, and then look at what Scenarios can offer.

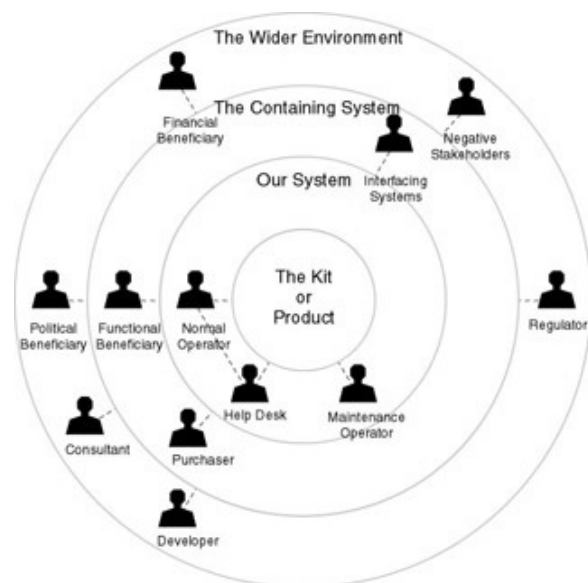
Stakeholders

Scenarios are for people: a way to explain what they want, and to check that what is being asked for is correct. People with a legitimate interest in a System are its Stakeholders. Who are they? At the least, they represent diverse groups.

Incidentally, scenario techniques such as role-play can be used to document stakeholders' viewpoints: they can tell the story of how people behave and to paint a picture of their “motivations, expectations, and attitudes.” Such an ethnographic perspective—getting a user experience point of view—is valuable, as it gives engineers the chance of “putting on the shoes of the people you've created, and looking at their problems and your solutions through their eyes.”

While this may seem a surprising application of scenarios, making a user profile is common practice in the worlds of user-interface design, marketing, and product management: perhaps it should be more widely used in system design also.

This diagram shows an 'onion' model of a system's stakeholders. Since the concepts of both 'system' and 'stakeholder' are slippery, let us take a moment to define what we mean. The central circle, labelled 'The Kit or Product' denotes whatever it is that we are making—the hardware and software, the equipment, machinery, and installations of any kind that we are specifying and designing. If we were making something for a market, we would call it our Product. That is often called a *system*, but we'll choose to call 'Our System' the next circle out. 'Our System' consists not only of kit, but also of the people who are actually operating that kit. Consider an aircraft: it's clearly a system, as it consists of a set of connected parts that together produce the behaviour of controlled flight. Those 'parts' include engines, airframe, controls, electronics, and the aircrew. From this point of view, therefore, any Stakeholder who is directly involved in operating the Kit or Product is part of Our System, the one that produces the results we are describing with our Scenarios.



The onion model provides generalised ‘slots’ such as ‘Normal Operator’ and ‘Maintenance Operator’, which need to be filled with specific stakeholder roles for a particular project. For instance, the Normal Operator slot in an aircraft contains roles such as the Pilot, the Navigator, and arguably the whole of the rest of the aircrew. The Maintenance Operator slot for an aircraft includes the first-line daily maintenance crew—the refuelling team, the cleaners, and so on, as well as second- and third-line roles like radar fitter, engine fitter, and so on. All such ‘direct’ Roles, those that belong in ‘Our System’ slots on the onion model, can play Operational Roles in Scenarios, and can serve as ‘Actors’ in Use Cases.

All other Stakeholder Roles, those outside the ‘Our System’ circle, are ‘indirect’ or non-operational; these terms have some value, but they cover many different kinds of roles. For example, the Aviation Authorities who certify that an aircraft is safe to fly belong in the ‘Regulator’ slot in the ‘Wider Environment’ circle. Regulators don’t pilot planes; they don’t even work for airlines that operate those planes; but although they are outside those ‘system’ circles, they are among the most important stakeholders in an aircraft development project.

Finally, notice that the owners of Interfacing Systems are also Stakeholders in our system: they are part of the ‘Containing System’ that includes our system. In the case of an aircraft, that Containing System consists of the airports and air traffic control that the aircraft operates with. For example, the towbar that connects a tractor to an aircraft’s nose-wheel is an interface: both tractor and aircraft have to comply with a standardised design so that they can be sure to be compatible. Interfacing Systems (like tractors) can appear in scenarios and can serve as Actors in Use Cases, even though they are not human. But while the company managing the towing of an aircraft is potentially a Stakeholder, tractors, and towbars are not.

Story

The story form is, as has been argued above, as old as language. It seems simple, but is hard to describe. A story is essentially a narrated description of a causally connected sequence of events in some domain, or more usually of actions taken by a small number of interacting protagonists:

A poor woodcutter had two children. There was little to eat, so to keep them safe he decided to send them to his brother, who lived deep in the forest...

The pilot has no stick or rudder pedals. Instead, a small joystick mounted in the armrest allows the pilot to control the craft with one hand. With the joystick in the neutral, vertical, position, the craft remains stationary. When the pilot tilts the joystick to the left, the craft turns, banking according to the joystick’s left–right angle to the vertical. When the pilot pushes the joystick forward, the craft accelerates smoothly...

Even the simplest story has the power to create in our minds an image of a world—maybe magical, maybe technical—so that we almost feel we are there. Questions and guesses arise unbidden. Will the children meet a witch, or an ogre? Can the craft fly or hover? If it does, can it climb or descend vertically, and if so how would you control that with a little joystick?

The most basic form of story is a spoken or written narrative, but stories can also be told more visually. Indeed, story underlies nearly every kind of scenario. For example, an organisation could make a film of people actually using a legacy system, set in context by shots of customers phoning up, asking for a service, and so on. More simply, a story can be told by acting out a scene, live in a workshop or recorded on video. For instance, a sub-group of workshop participants can pretend to interact with a future device, to illustrate a desired capability or to bring to life a possible problem that ought to be handled.

Stories bring with them a wealth of context, mostly unwritten, from our shared culture. As long as this is what the story’s author intended, it is useful, as it provides a frame of reference in which the reader can evaluate the story as a whole, and make sense of its individual statements—which can therefore individually be short.

Stories provide an internal logic—of a sequence of events in time; of causality—which is valuable to engineering as it permits, indeed encourages validation. Questions like

- *Are these steps in the right order?*
- *Has a step been omitted?*
- *What could go wrong here?*
- *Is this story complete?*
- *Would doing this achieve the goal? Can this actor carry out this action? Is this a safe approach?*

spring naturally to mind, and stakeholders—people with a legitimate interest in a system—in practice correct stories rapidly and deftly, in marked contrast to their behaviour when asked to review a list of requirements.

Stories are coming into favour in various branches of engineering, notably in human–computer interaction and in ‘agile’ (or ‘extreme’) methods of software development.

Development based on stories alone, is possible only if you are not too concerned with perfection (at least at the start). If you want a system to be as safe as possible, you will specify and analyse its every behaviour, and use formal logic to prove that its code is logically equivalent to its specification. If, frankly, you don’t care but you have a few situations in mind that describe what you’d broadly like the system to do, then telling a few stories may well be quite adequate—the team can make the system converge with stakeholders’ intentions by iteration (rapid prototyping, agile methods, etc.). These mindsets are poles apart, and it is easy for engineers with such different backgrounds to talk past each other. However, there are more structured forms of scenario that many engineers may find more comfortable.

Situation, Alternative World

To the businessman or politician, a scenario means a projected future situation or snapshot. Typically, either a team brainstorms a few possible scenarios, or more scientifically, a simulation is run say, 15 years into the future to project a set of variables.

For instance, in an approach pioneered by Shell, an oil company might look at a moment in the future where the oil price is \$100 per barrel, alternative sources of electricity are cheap at just 10¢ per Kilowatt hour, and fuel cells are available for 50% of all cars. The values of these variables suggest a pattern of energy use very different from what would happen if electricity cost \$1 per Kilowatt hour, and so on; and that pattern can be convincingly modelled using suitable mathematical models and simulation tools.

This usage of ‘scenario’, as an instantaneous snapshot that illustrates the outcome of a set of trends, is plainly helpful in long-term business planning. In a world where politicians had any foresight, it would undoubtedly be useful in long-term government planning also. But on the whole it seems to be rare in systems engineering. When engineers need to refer to a specific outcome they tend to use other words, such as system state or failure mode.

Simulation

Simulations can, as we’ve just explained, be used to model more-or-less static situations, but they are also ideally suited for exploring dynamic, story-like aspects of scenarios.

Any story that has a direct concrete interpretation can be modelled, animated, and simulated. If stakeholders say they want to run trains with more passengers on board, and that the trains must stop for half the time at each station to improve speed and punctuality, then engineers can model how long it would take to get so many passengers onto and off the trains given different door sizes and carriage layouts. Consider the following scenario:

A train crush-loaded with 2000 passengers pulls into Central Station in rush hour. There are 1000 people waiting on the platform. Seven hundred passengers leave the train and 700 join it. The driver closes the doors and the train moves off 30 seconds after it halted.

Simulation can give precise answers about whether such a scenario could be realised with any plausible design, given the existing constraints on train length, platform width, and so on. Simulation can also be used to evaluate the implications of alternative possible worlds or situations (see above). Used for that purpose, simulation is interesting not so much for its ability to step through a story as for the results that it generates (e.g. did the passengers manage to leave the train within 30 seconds?).

The sorts of comments and suggestions that stakeholders typically make when they first see a running system or animated simulation are ‘But what if ...’ or ‘And then I’d like to be able to...’. These illustrate why scenarios are so valuable: once the implications of a set of choices are displayed, everyone can see if the result is right or needs improving.

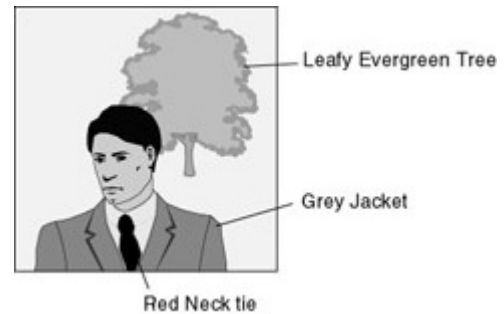
A simulation model can show stakeholders the likely effects of their requirements when implemented as design parameters. We may be interested in the dynamic results, as when people queue for admission to a stadium and its facilities at peak times, or in a more static conclusion that option A is more cost-effective than

option B. Either way, appropriate use of simulation—possibly involving many runs—can explore scenarios that might be too slow, too costly, or too dangerous to try out for real. Simulation also answers questions too difficult to visualise and to make decisions on without the kind of quantitative evidence that simulation can provide.

There is a close connection between scenario, simulation, and prototyping: all explore ways a system would work when in service, and all imply the need for an iterative and participative development life-cycle.

Storyboard

A variation on the story is the storyboard, invented in Hollywood in the 1920s and still in wide use on film shoots of all types today. Anyone who has tried to understand a play by reading its script knows the problem: the words as they lie there on the page just fail to conjure up a visual impression of the action. The scripted dialogue is a kind of scenario, but while it is essential for telling the actors what to say, it is not sufficient for organising shoots. If the heroine walks out of one shot in a blue skirt and a red top, and into the next in a red skirt and a blue top, the continuity of the story is badly damaged. The team producing the shot needs quite detailed information to set everything up correctly, to ensure continuity among other things. Each frame of the storyboard gives just the right amount of detail for that shot; the storyboard as a whole tells the story of the film.



Unsurprisingly, storyboards find their main engineering use in human–computer interaction (HCI). HCI designers may sketch all or some of the desired screens in the user interface. These screen concepts can serve as prototypes: they are effectively software-free implementations of the system in various states. A user-interface prototype acts as a window into the (future) system, allowing stakeholders to reason about its behaviour.

On one project, when the user-interface requirements were extremely uncertain, I made a set of storyboard sketches in ink and watercolour, before going on to use a sophisticated GUI design tool. Colleagues later joked about my Early Watercolour Phase, but at least a dialogue on the required functionality and design had been started. The design tool, incidentally, did a beautiful job of setting up a possible graphical design, but it was a dangerous journey down a ‘rabbit hole’, as it focused my attention too sharply on the implementation of the interface—when what the interface ought to have been showing was anything but decided.

Sequence

A straight-line sequence of (possibly numbered, typically interactive) steps taken by independently-acting (presumably intelligent) agents playing (system) roles is roughly what most people mean by Scenario in engineering today—but note all the caveats. Synonyms or partial synonyms for this include Operational Scenario—itself part of a Concept of Operations, (Test) Case or Script, Path, and Course (of actions or events). It is evident that the terminology is slippery. We’ll try to define Scenario a little better below.

Given that systems are by definition, complex structures made up of interacting components, there are usually many possible interaction sequences for any system. For instance, if a tool offers only 10 menu options that may be chosen in any order, then a sequence of just three commands can be constructed in 1000 ways, or 720 if repeat commands are not allowed. When a tool has over a hundred menu options, as does Microsoft Word, and many of those allow numerous choices and sequences are typically many commands long, the number of scenarios that could be generated quickly runs into the billions.

This is at the heart of the twin problems of specification and verification: it is impossible to test everything. There are simply too many possible test scenarios. But by the same token, it is impossible to specify everything either, if we attempt to do it with scenarios alone. Surely it would be better to rely on analytic, program-like structures that can in the twirl of a *while* loop or a *case* statement take care of many possible scenarios?

Well, it would be much better, save for one crucial fact: humans are not at all good at expressing themselves in precise, systematic, logically complete, and consistent form—or in understanding such things either. Even

professional programmers make frequent mistakes when encoding system behaviour in software. It is said that carefully debugged code still contains 1 error per 1000 lines. We should expect specifications written like code to be just as error-prone—otherwise, we could solve all the problems of software just by compiling code from our specification languages.

Since requirements come ultimately from people, we have a problem: we need precision in requirements, but we need people to express and understand and agree the same requirements. Precision means analysis and formalisation by engineers. But requirements elicitation and validation must be from and by stakeholders—people whose expertise is not in stating requirements. Therefore an altogether softer approach is mandatory.

Scenarios offer some hope of bridging this particular gap, as they are good raw material for analysis, but are close enough to ordinary stories for people to understand without effort. There are many possible ways in which straight-line time-sequence scenarios might be applied in engineering. Notably, they could be either inputs received from stakeholders, or ‘people-friendly’ outputs generated by a specification engine (presumably one that contains a database).

Any bridge between the ordinary world and the domain of systems analysis is subject to enormous tension. Research engineers of different kinds constantly come up with ways of ‘improving’ the languages and notations used to express requirements and specify systems. The naivety of a simple scenario is a red rag to a bull to people thinking about how to make better specifications. Obviously you can just add a branch here and an exception mechanism there and a looping construct down here, and in no time at all you’ll have a fully capable specification language! Ah yes, but you’ll also have left your stakeholders stranded on the far side.

Numerous researchers of undoubted intelligence claim that they have successfully demonstrated that a group of normal ‘users’ readily understood some (usually graphical) notation or other. Well, maybe they did; but the users were buoyed up by the excitement of being pioneers; they were probably self-selected as being the most adventurous people in their company—since they’d agreed to work with a researcher!—and not least, they had the pioneering researcher playing the role of the humble systems analyst. It isn’t surprising that these approaches often don’t work in less rarefied atmospheres.

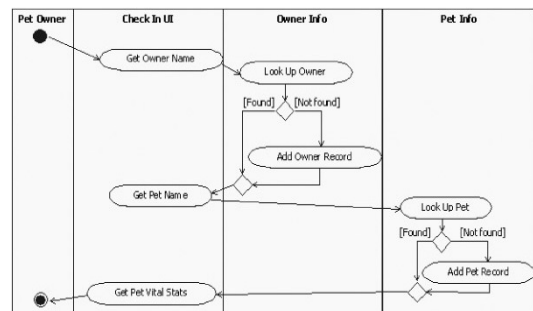
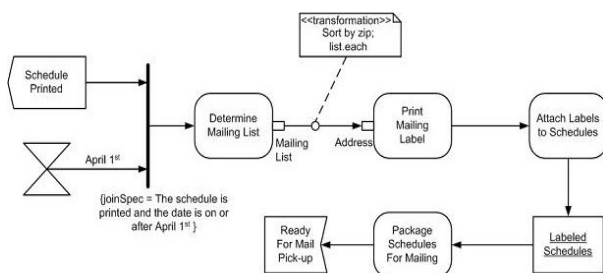
The plain old list-of-steps scenario is not a perfect medium for specification. But it is a workable medium for the purpose of getting started on the task of writing requirements, and providing grist for the analyst’s mill. What is incontrovertible is that scenarios can be understood by ordinary people and engineers alike.

Structure

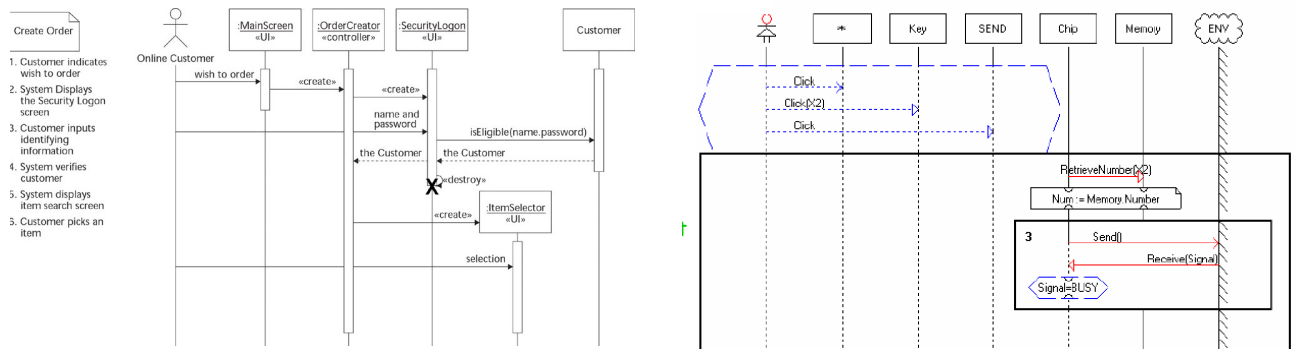
The further you go into analysis and specification, the greater is the pressure to impose structure on scenarios.

There are some longstanding techniques for defining the order of steps in a business process or program.

- The flowchart, despised by analysts but beloved of businessmen, is alive and well even in the UML as the Activity Diagram. Flowcharts give a good impression of what has to be done when, including places where decisions are required and the outcomes of those decisions. What flowcharts, on their own, don’t do is to show who is responsible for each action. This is quite a serious weakness, given how central stakeholders are to requirements. Happily, UML has also adopted the swimlanes diagram, which has been in use since the 1920s. Each role (or ‘actor’ in UML) has a lane. Roles carry out actions by swimming down the page. Combining an Activity Diagram with a background of swimlanes allows the humble flowchart to show clearly who does what.



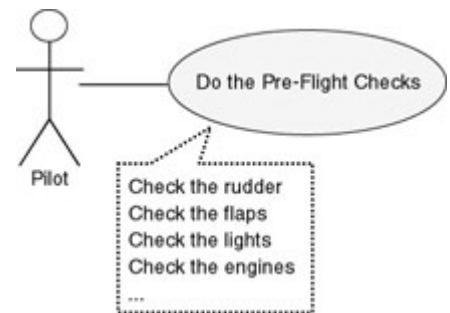
- Another notation that can be used to analyse scenarios is the (message) Sequence Chart, which is found in the UML and the ITU frameworks. This provides a timeline for each object or role down the page, with messages between roles shown as labelled horizontal arrows. Various different symbols can be used to annotate the sequence to define more elaborate behaviours, for example, an hourglass to denote a timer for time-triggered events. It seems clear that such notations are suitable for analysts rather than for scenario-like communication with other stakeholders. A yet more sophisticated notation, the Live Sequence Chart, takes sequence charts even further into the analyst's world.



Activity Diagrams and Sequence Charts may be fine as representations, but they do not solve the problem of identifying all the if..then..else.. branches—all the exceptional business events that have to be handled. There are usually far more of these than anyone thinks of initially. Programmers are all too familiar with filling in all the missing elses by experience and guesswork.

UML's answer to this problem is the Use Case, which has evolved far beyond what was originally proposed. The Use Case began life as not much more than a Story or Sequence: it had a name, an actor, and some text.

"A use case is just what it sounds like: it is a way to use the system. Users interact with a system by interacting with its use cases."



A use case diagram is only a summary, but unfortunately, several tool vendors eagerly jumped on to the bandwagon and created tools that concentrate on drawing Use Case diagrams, leaving aside the much duller work of dealing with what the Use Cases contain—scenarios. This would be funny if it had not misled so many people. It is a necessary step to identify the various ways that people and systems may interact, but the scenarios give much more information.

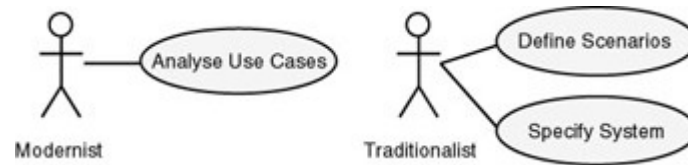
Erring the other way from such a diagrammatic over-simplification, several authors have suggested ways of adding more structure to the contents of each use case, and to the relationships between them; and this trend is continuing.

A Cockburn Use Case has a Goal—its title: a very sensible minimalist choice; a Main Success Scenario composed of about three to nine steps; and usually several Extension Scenarios to describe other things (such as failures) that can happen, and must be handled. In addition, Cockburn provides slots for a range of other requirement-like things including Preconditions, Stakeholders and Interests (i.e. Viewpoints); Minimal (i.e. failure) and Success Guarantees; Scope and Level. To fill in all of this is a substantial piece of work, but in return you communicate much more of the stakeholders' intentions with far less ambiguity than you would with just a single scenario.

A closer look at Cockburn's approach will help us towards an understanding of what scenario means. Notice that the term 'scenario' in Cockburn's usage has subtly shifted its meaning. Consider what happens if you start executing a Use Case and a failure occurs. You execute part of the main success 'scenario' and then the appropriate failure 'scenario'—but the entire sequence of activities you've just executed is what we'd call a scenario: **a particular complete path in time through the options available when interacting with a system.** The Main Success Scenario, which at first sight matches our initial idea of a scenario, turns out to be just a tree-trunk, with several branches documented elsewhere. A Use Case changes from being pretty

much 'a named scenario', to an analytic description of what happens when one or several actors try to achieve a goal with the system.

Now analysis is necessary for system specification, but a key question is whether it should take place in a scenario / Use Case framework, or somewhere else, with reference to scenarios which themselves can remain simple and comprehensible. I'll call these the Modernist and Traditionalist positions, respectively.



The Modernist view is broadly that if you know what your stakeholders want as a set of Use Cases, you can build your software straight from that: Scenarios are Specifications; the Pattern of the Problem corresponds to the Process of Design.

For example, to develop an order processing system, you identify Use Cases such as Place Order, Make Payment, Deliver Goods and so on. You then detail what is meant to happen in each case, identify what can go wrong and work out what should happen in each of those extension cases. You add any other details such as guaranteed performance and availability, and you are ready to start designing the software.

The Traditionalist view is roughly that scenarios are fine for describing how your system may be pressed into service, and that they may well suggest a range of functions and quality requirements, but that they in no way express all the requirements, let alone specify the system.

For instance, if your tanks are going to have to drive across the desert, they must not sink into soft sand, and the engine and other parts must not get damaged by ingesting sand. These needs may translate into qualities such as low ground pressure (below so many KiloPascals) and functions such as the ability to filter sand-laden air. But whatever the ultimate analysis of requirements, these are entirely separate from the scenarios. It would be quite inappropriate to expect the soldier who describes what he wants to do with the vehicle in the desert to know about KiloPascals—indeed, if he is only describing desired actions, he knows neither the area of the tracks nor the weight of the vehicle, and sand mechanics isn't his forte either. The scenarios are insufficient as system specifications. But the scenarios remain valuable for discovering requirements, for acceptance testing, and as something to validate the requirements against—so traceability becomes essential.

TYPES OF SYSTEM

It is practically a hopeless task to describe the types of system where scenarios may be used. There are no obvious limits to the applicability of scenarios. This is not to say that every development will find scenarios useful, but simply that there is no a priori reason to believe that scenarios are inappropriate to whole classes of development project. Projects should consider writing scenarios as a matter of course.

Development projects are those that are doing something new, and therefore something risky and uncertain. This is the natural home of the scenario, as discussed earlier in this chapter. It matters little whether the project concerns hardware or software, structures in concrete or structures in silicon, complex business interactions or home entertainment, deep space exploration or easier-to-use video recording.

Projects that are essentially routine—doing yet another inspection of a civil engineering structure like a bridge, installing yet another virtually identical computer network—are not developments in this sense. Scenarios will only be useful to such projects individually when there are local conditions that have not been encountered before; then, there may be value in examining threats to the project, and planning suitable mitigations. On the other hand, there may be a powerful case for preparing a set of generic scenarios to be applied to all bridge inspections or all network installations.

Scenarios can be used in engineering of all kinds, not only software. However, software is today extremely pervasive. It is finding its way into many structures and devices that were formerly completely passive—bridges have inbuilt monitoring systems; binoculars have image stabilisation; keys and credit cards authenticate themselves using strong encryption.

Much of the complexity of systems now goes into control software—in aircraft and luxury cars, software is already the single most costly constituent. DaimlerChrysler predicts that by 2010, 45% of the cost of a Mercedes will be in electronics—what in aerospace is called avionics, combining both hardware and software. This has profound effects, some beneficial, some not. The complexity, size, and weight of hardware components is falling, and the chance of hardware failure is therefore falling with it, so we can build systems that are smaller, more efficient, and more reliable than ever before. But taking onboard software into account, we are also building systems that are growing rapidly in complexity, and we are in danger of introducing a multitude of unwanted interactions and failure modes. Overall reliability and safety are not guaranteed to improve in this situation. They will only do so if we keep control of system behaviour. Scenarios are valuable tools in this struggle, because they contribute in several ways to our understanding of wanted behaviour, and to our ability to prevent unwanted effects. This is the reason why DaimlerChrysler (for instance) is looking at stories and use cases. Obviously, many other tools are needed to maintain control throughout the system development life cycle, as requirements are passed to subcontractors, and design choices introduce additional interfaces and risks.