ATAC

ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Support from testing for fast and dynamic software development

Project number:	ITEA 2 10037
Edited by:	Matti Vuori, TUT
Date:	2014/11/06
Document version no.:	*1.0*

Table of Contents

1. Introduction

In ATAC we have researched test automation, but it is not an island. It is used in the context of the overall testing activities, which is why we always need to give a context and frame for it – especially when the issue is using it in practice and not just developing test automation technologies. Here the framing is:

- Product development.
- Everything that done for product quality (and that in a broad sense, meaning its success factors).
- All testing related activities, that is: all activities that produce timely information to support decision making and development activities.

Today's product development needs to be fast and dynamic. But how do we manage quality at the high speed? A company should only try to drive at a speed it can handle. There is also not only one type of speedy product development – there are many ways to do that. In this short document we will take a short look into what kind of testing the various types would require – or benefit from – and give some additional ideas for handling speed with managed quality.

2. General aspects of speed

2.1. There are many kinds of speed

What is "rapid" or "high speed"? We know that terms can often mean various things and things can have many metrics. Now, whenever fast product development is mentioned, it also can mean several things, some of which can have very different characteristics.

The following are some of the types of speed in product development.

Fast continuous speed

- Fast product pipeline. New products are brought to market at fast (but sustainable) pace.
- High continuous speed of producing new value to the customer. For example new features are published "constantly".
- Continuous deployment. Regular updating of product in use, for whatever reason. Short time from entry to process to exit.
- (Note that the velocity of output does not necessitate that the time spent in development is short – as analogy consider a highway that can be hundreds of kilometres long, but "outputs" cars every second.)

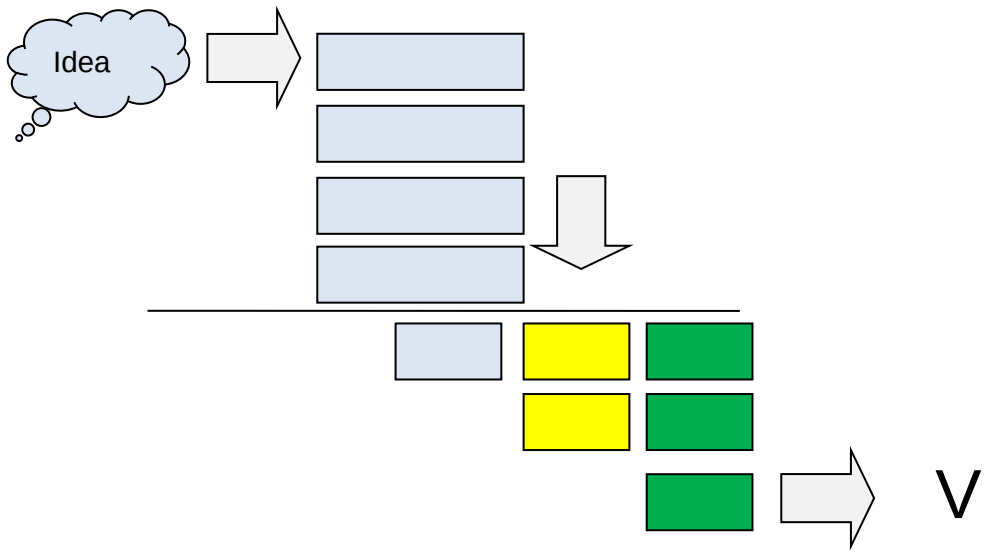


Figure 1. Flow-based new value introduction.

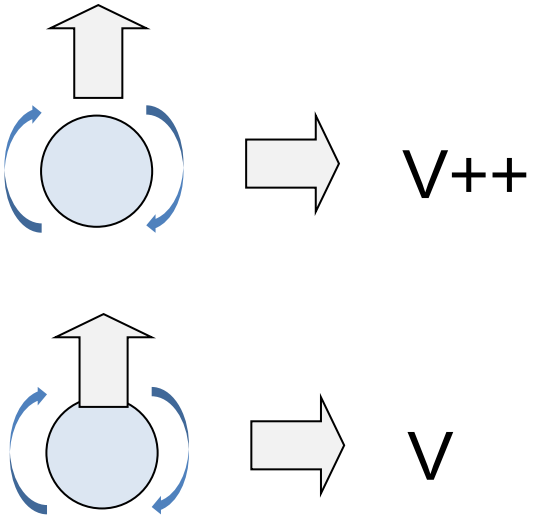


Figure 2. Rhythmic new value introduction.

Fast projects:

- Fast time to market. The time from idea / concept to market entry is short, meaning that the product development is done at a fast pace.
- Fast velocity in large project keeping their schedule reasonable.
- Rapid pivoting of a product. If a product needs refocusing, it is done rapidly.

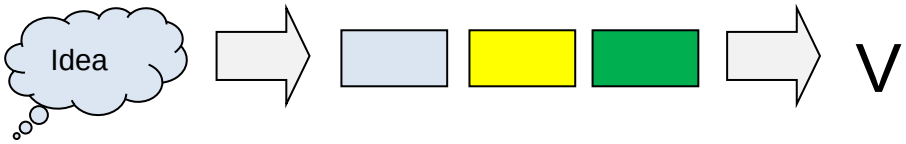


Figure 3. Fast project.

Fast response:

- Fast response to any action from competition. For example when a competitor publishes a new feature, own product is updated to match that rapidly. The “browser wars” between Netscape and Microsoft in the 1990’s is an example of that and really emphasised high speed in software development for the first time.
- Rapid reaction to emerging needs. When a need for a new feature emerges in the clientele, it is met with rapid updating of the product.
- Rapid response to problems, threats, risks. Rapid bug fixes and hot fixes.
- Rapid response to changes in collaborating systems, including other systems in the overall architecture, social media APIs.

Fast change of platform or ecosystem:

- Fast introduction of products to another operating system or device platform (besides old platform – that is, turning into multi-platform provider).
- Fast porting of products to other environments.
- Fast changing of primary platform – abandoning old, embracing new.

So, there clearly are different types of speed and they require different competences: the ability to have a high average velocity is a very different skill that the ability to take a spurt as needed. Also, it can also vary whether the process is dominated by rhythm or more linear logistics.

2.2. What is the optimum speed?

The optimum speed varies from viewpoint:

- For customers, the optimum speed is what suits them best. If things work and there are no unfilled needs, change is not wanted as it causes work and potential problems.
- For vendor, the optimum speed is what maximizes business success on a chosen time span. That includes factors such as sales, costs and risk level.

In product development, all new features must be such that the customers notices them and notices the value provided – after all, what is the point otherwise? Building engagement, interest and temptation is even more important than producing “value”. Very small incremental changes are not noticed at all! So increments should often be grouped so that the value is apparent – customers see it, it can be used in marketing. And testing should be able to show the magnitude of improvement – use some metrics to show how much better the new version is compared to the previous one.

A myth about speed of market entry is that one should be the very fast in entering a new market or bringing a new concept to market. Often it the case that the last on the market wins… if the entry is done before someone manages to build a dominant market position. This is due to the difficulty of selling a new concept: it is hard for customers to make buying decisions when there is nothing to compare the product with. For later market entries it is possible to compare and select the “best” one – best usually meaning the most desirable. Also, the later entrants are able to develop their technologies better and learn from others’ mistakes.

Testing should, obviously, help companies understand how their product compares with the competition.

Of course, the need to provide value fast depends on many factors, such as the competition dynamics and the nature of the new value. When the new things are a “must”, they need to be delivered, or customers will get unhappy. If they are optional extra, they are either something to sell, or something to increase customer happiness and also to provide a symbolic note about activity and constant consideration about customers’ needs. But they can also be something completely new, something unexpected, and in that case rapid “business as usual” delivery can be a mistake. Instead, release should be planned carefully so that it is communicated well and differentiates from other activities.

2.3. Things speed requires

Speed usually requires things like this:

- Very skilled driver.
- Control and coordination in vehicle and in environment.
- Very reliable systems – processes, collaboration.
- Clear rules that everyone understand and adhere to.
- Very powerful and reliable brakes.
- Seat belt and airbag.
- Plans for emergencies.
- Goals milestones, a good track to proceed on.
- Knowledge on where we are now.
- A speedometer – metrics.
- Sense of time, rhythm.
- Visibility to others about where we are speeding.

Of those, testing will provide brakes – it can help the project to stop before crashing into something. It can also provide knowledge where we are now, which aids in control and sharing the status with the environment.

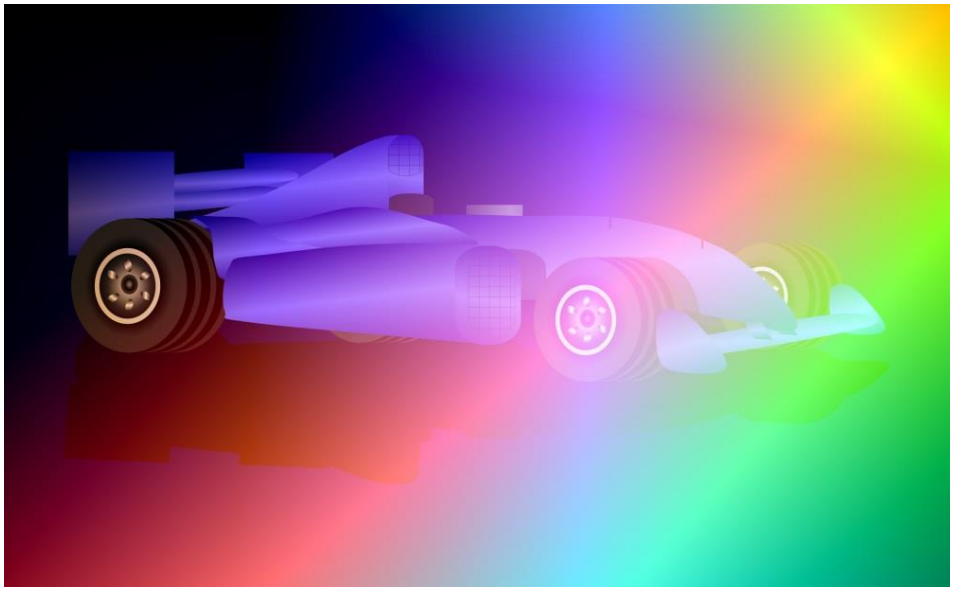


Figure 4. Speed and tools for it can be tempting but dangerous.

2.4. Risks of high speed

Maximum speed is not optimum, if it is compromises the business mission. Buses from Tampere to Helsinki could be faster if they didn't take any passengers. High speed does produce benefits, but there are also risks in it. These are some examples of risks:

- Continuous updates lower the desirability of new product versions, making it harder to market and sell them.
- If speed is not managed, all forms of quality can suffer.

The latter is due to processes not being developed suitably for the speed or there not being sufficient resources. Examples:

- If testing is not good enough, new product releases can contain defects. (Luckily, they can also be pulled back or repaired fast.)
- Trusting too much on test automation can let defects slip past that would have been caught by manual testing. Or, if automated regressions testing is not sufficient, trivial regression failures can happen. For higher development speed, both automated regression testing and manual testing need to be improved.
- If the product architecture is not designed for agile changes, it will rot and the product will be harder and harder to maintain.
- If the new features just cumulate and add to the feature count, the product will gain complexity and will be buggy and hard to maintain – just like any current "high end" product.
- If documentation processes do not have the necessary resources, documentation will lag behind, causing problems to the users. There are plenty of examples of this – for example, LinkedIn changes its functions often and the documentation often refers to functions that do not exist anymore.
- Quality of user interfaces will often suffer. Important functions are forgotten to include in new revisions and usability assessments and testing are neglected – not to mention good design of the new functionality.
- When startups evolve to a point where they start growing, gaining more speed in every activity is essential. If practices are not adjusted or changed to meet the new needs, problems will arise.

Clearly there are many risks in getting up to speed and various types of testing can be used to control some of the risks. But the most important thing is that speed is not the goal, it is just a tool for something – maximised customer satisfaction, keeping products ahead of competitors etc. And that is the biggest risk: trying to reach speed for speed's sake, without considering why it is done and what are the things that would cause the biggest benefits for business.

"Debt" is a word often used in this kind of situations. Existing technical debt and process debt hinder speed improvements and working at higher speed will adjust accumulation of technical debt. So in every speed improving activity there needs to be a focus to various kinds of debts.

One of the challenges is to be able to have the speed at the long term. At one phase of a company the focus of speed may be in concept level innovation, and in other phases in “feature logistics” – giving small new things to a varied clientele. Allocating the always too few resources to wrong things can be hazardous. For example, if a startup puts all its energy in the development of a multiplatform continuous deployment engine, it will have less energy to use in the critical task of developing desirable products. And good, fast processes that are in place, will tie the personnel into existing things, not allowing for mental renewal. That is agility gone horribly wrong.

One problem is that engineering mentality may start to drive business speed, not just enabling it. Speed of product development is a business issue at heart and thus needs dialogue from various viewpoints:

- Customer needs.
- Business opportunities.
- Product development and innovation processes. Testing as provider of information.
- Production capabilities and enablers.
- Quality management. Testing as provider of information.
- Risk management.

2.5. Deployable does not mean desirable

A mental risk is to see product development as “logistics”, as operating a value transferring machine. In that mind-set, speed records are broken what people think less about what is being produced – humans being humans. Testing gets narrower focus, moves from validation to verification. Just like when travelling fast one concentrates on checking whether we are on the map where we should be, rather than whether it makes sense to be on that road.

One must remember that software development is not serial production. Every new feature, every new build, every “item of value” is something unique and must be assessed carefully. There must be time for good validation, because it cannot be left for the customers. And it cannot be done in a speedy deployment pipeline, as it requires an integrated whole and peaceful mental environment.

Sometimes the levels of quality are presented that any new feature must pass, for example:

- The very first level is safe and secure. Nothing must be deployed to customer that is not safe and secure.
- At the next level is deployable. Something can technically be delivered to customer if needed. And that does not mean that it should be deployed.
- Third level is useful. The new functionality is that is in itself something valuable.
- The fourth level is technically solid. The new feature has no technical problems.
- Fifth level is usable. The new functionality can be used easily and without risks for its purpose.
- Next level is good user experience. While usable, the new feature offers good experience for the user.
- At the highest level is desirability, which combines other factors. When something is really desired, then the new deployment is seen to offer high value and is received with gratitude.

These (or equivalent) are the levels that should be assessed and testing should have a role in that.

Priorities on those levels obviously depend on the type of product, see table below. Of course the priorities vary case by case, so this is just a visualisation that aims to show what kind of characteristics should be designed and assured by testing and other means.

*Table 1. Priorities of quality levels for various system types (simplified). *** = critical, ** = important, * = relevant*

Quality level	Consumer product	B2B information system	Infrastructure technology	Safety-critical system
Desirable	***	***	***	
Good user experience	***	**		
Usable	**	***	*	***
Useful	*	**	***	**
Technically solid	**	***	***	***
Deployable	***	***	***	***
Safe & secure	**	***	***	***

2.6. Does more speed mean more work?

Higher average velocity obviously means that more is produced, meaning also more things to test. That could imply a need for more resources, but the speed could also be reached with current resources when things are done more effectively, with perhaps a leap in process capability.

More rapid reaction to needs and external events does not mean that in average more work is done. It is just done when it is needed, as fast as required.

Fast pivoting or fast concept creation is more a matter of product development competence and utilisation of technologies that enable fast product creation.

But the main issue in all the types of speed is that everything must be controlled professionally. Otherwise there will be chaos and lots of unnecessary work.

3. Consequences of going for speed

3.1. Traditional use of testing to gain speed

The traditional pathology has been to just skip testing or do less of it to gain speed. When most testing has been left to the end of a project and there is a hurry to get the product out, testing was just skipped or done less.

Of course, nowadays most people understand that testing must not be left to the end of the project, but should be started in the beginning and done continuously with an aim to keep the product technically stable all the time – ready for deployment, stable to handle any changes without falling apart.

But healthy principles include:

- Test in parallel. Let testing proceed at the same time as development.
- Test less. That requires less implementation, a more focused product, which may be desirable. One development principle is to “develop for now”. That means that testing should focus to most important issues in next release – yet at the same time to keeping the platform robust for any new developments. Good up-front work in user interfaces reduces need for testing later (but does not remove it).
- Test faster. Use automation and competent testers.

- Test in advance. Use well-tested components.

3.2. Need for many layers of thinking

Testing is still dominated by “technical testing”, that is, functional testing and similar. While it tackles business and user processes and tasks, it is still on the engineering level. Consider this statement from a recent testing magazine:

“A program is testable if and only if its Turing machine representation can be flowcharted, is deterministic and is always-halting, and all possible faults can be sensitized to output.”

It demonstrates much of the underlying thinking in testing. At the level of product development one could say:

“A product is testable anytime when it can be presented in some way”.

Testing and its ideal are seen to be part of the engineering paradigm. But so is the general product development culture! Only gradually are we learning that engineering is not product development – we need a broader view to the systems under development and also to the needs of the customers. An eye-opener was at a time the understanding that paper machines need to be designed to be attractive. Similar learnings will happen in many domains today.

Of course, we need many layers in testing thinking and action, all of which fulfil their particular needs. The problem is when testing is too concentrated on some level, ignoring others. The table below visualises the thinking differences between engineering and product-oriented testing.

Table 2. Comparison between engineering-oriented and product-oriented testing (caricature)

Element	Engineering-oriented testing	Product-oriented testing
View to system	Internal view, structure, functions, architecture	External view, customer value, requirements
Metrics	Absolute	Relative to competition, past
Source of quality criteria	Standards, own views	Customer desires
Compromises	Coverage	Technical versus filling needs, including release speed
When testing is done	When closing criteria are met (such as coverage)	When it has found the information needed for decision making or activities
General testing basis	Specifications	Reality, customer’s world
UI testing basis	Usability, standards	User experience, reaching goals, usability
Tester focus	Defined testing tasks and methods	Information provision using any means
Process approach	Testing lifecycle	Agile response to information needs
Role of test automation	Should test “everything”	Frees testers to use their mind for finding new information
World view	Testing creates control	The world insecure

Still, it must be remembered that testing style and culture are always a reflection of development style and culture and need to have a suitable match. Developing of testing requires developing of the whole product development activity.

3.3. Mental landscapes and competences

Doing things rapidly means that there is not much time to ponder and discuss things. Developers and testers must have clear understanding of critical issues and a risk-aware mind-set. In fast workflows we see in practice the principle in Toyota production system that everyone may stop the process at any time – for example, a manual tester must have the courage to say that this build is not good enough to be deployed to customer.

All modern testing requires understanding of the business and the customers' needs and this "domain of speed" emphasises it.

Strictness is needed to keep the processes rolling and that applies to testing too. Tests must never be skipped. One good feature of automation is that one can never think that "this is just a simple one-liner and it doesn't need not be tested" – it will get at least some automated regression testing automatically. And when manual testing is integrated in the workflows, it will be (or at least should be) signed off by a manual tester. Of course, this is how things should work, and not how they always work.

One needs to differentiate the constant throughput and the time spent on development. Which one is more important, needs to be analysed and things balanced.

More essential than speed of a train is that you get to the next station without crashing…

One needs to remember that speed is not a value as such. Number of deployments per time unit is no value as such – or rather it is waste. Value for business is the goal.

Table 3. Some essential tester competences for two main types of speed (simplified)

Competence	High velocity	High reactivity
Tester identity	Strong tester identity as counter force for logistic thinking	
Personal strength	Courage to stop deployment if quality is not sufficient	
General competence	High generic tester key competencies	High generic tester key competencies Large personal "toolbox" with which to tackle fast any new testing task Ability to work fast and in agile manner Exploratory testing skills
Risk related	Regression awareness	Ability to analyse of how changes affect system Regression awareness Risk analysis skills
Understanding customer		Understanding how changed thing will be used – and how it should be tested

Competence	High velocity	High reactivity
Test system control	Configuration management skills	
Programming skills	Varies. Scripting skills very useful.	

3.4. About methods and tools

Sometimes misguided managers may think that testing needs not be that good, because faulty functionality can just be pulled back using the deployment mechanism. Such managers need to remember that what is ok for non-business-critical systems is not usually ok for customer solutions. But that emphasises the fact that the deployment mechanisms and the pull-back mechanisms must be very reliable – they need to be well designed and tested.

All in all, reliable workflows are essential, so planning and designing them is essential.

Many textbooks – and many "experts" – forget largely about manual testing, but for most systems it is a critical step before deployment. Functional testing should usually be done, but testing of user experience and usability must not be forgotten. Related to this is the idea that every tester should have programming skills – and use them daily. In fact, one could think that the culture of speed may benefit from people who have other approaches than test automation. Speedy business and good testing in general benefit from diversity in skills, approaches and thinking.

When going for velocity, there is a tendency to make "batch size" smaller. It makes testing easier as testing of new value can more concentrate on a manageable set of functionality. Instead of a release candidate for a new system version, it is just a matter of handling a small individual change (and its associated, hopefully small regression potential).

When the speed is about reacting to events, preparation helps: good architectures that make modifications easy; testing of upcoming changes in external dependencies in test environments etc. Not everything must happen in real-time.

Speed in any form requires good configuration management and that greatly helps in making testing more managed.

3.5. What kind of speed do you prepare for?

Consider the world of cars. A Formula 1 car is very fast, but hopelessly complex and only suits a given type of track. You can make speed records with that.

But how about the need to adapt to new kind of territory, new kind of business, new customers or new concepts? A rally car is frighteningly fast too, but can be used on asphalt and bad gravel roads and it is way simpler to implement and use. You won't make absolute speed records with it but those are not your business anyway.

So, development and deployment systems can become locally optimised. They may maximise the speed of technical deployment but not be optimal for speed in business level. Flexibility and simplicity in every area seems essential. That means also that all testing arrangements should support flexibility – or rapid agility:

- Capability to test new business ideas.
- Capability to test things at concept level.
- Capability to test technologies that are not good for automation (yet).
- Capability to test changed product technology, components and architecture.
- Capability to support experimentation at all levels – product concepts, technology trials.

- Capability to provide information that business needs for business decisions.
- Etc…

3.6. Catching problems fast

Systems aiming for velocity are based on workflows and logistics. With those, it is essential to catch any problems as early as possible. That means that for the technical quality the QA system should:

- Have good unit testing and static analysis.
- Have solid regression testing at all phases.
- Utilize tools such as static code analysis to pinpoint potential problems.
- Use test environments that are “exactly” similar to any production environments.
- Generally emphasize the front end of the process.
- Etc.

4. Testing challenges in various types of rapid product development and solutions to tackle them

4.1. Introduction

There are, luckily, many possibilities in testing that can be implemented to make the raising of speed possible. Here are some examples of them.

4.2. Meeting general challenges

Table 4. General challenges and corresponding testing solutions

Type	Special challenges	Specific solutions	Notes
(General)	Handling regression	Good regression testing integrated into the build process Exploratory testing for regression too Identification of affected areas (functionality, modules, code) to guide focused testing	Tools can be used to identify the effects of changes and regression tests can be targeted accordingly – from wide and shallow to more focused and deep
	Robust, stable code	Low level (unit) testing practices	Stable base is essential
	Assuring maximum value	Risk based testing Solid testing for things that produce new value	Testers too must understand where value comes from

Type	Special challenges	Specific solutions	Notes
	Fast deployment of test systems	Programmatically generated and configured test systems Programmatically generated and configured virtual machines Use of cloud services	Virtual machines become easier and easier to create and deploy
	Optimising of testing resources	Risk based testing strategy Re-usable test assets	Critical for large and complex systems
	Flexibility, adaptability	Competent testers Flexible testing tools Tools that need minimal adaptation and configuration Platform-independent (agnostic) tools	This is a key element of Lean – ability to immediately adapt to anything Physical testing robots are examples of UI agnostic tools
	Good testability to allow rapid testing	Testability design Choice of technologies Testability reviews	For both manual and automated testing
	Automating UI-level testing when testability of applications is bad	Testing tools that require no instrumentation in SUT (using shape recognition, OCR), such as physical testing robots	

4.3. New product development

Table 5. New product development challenges and corresponding testing solutions

Type	Special challenges	Specific solutions	Notes
Concept testing	Rapid testing at all relevant levels	Agile testing Flexible test systems	
Fast MVP-based first product development	Finding product preferences	Preference testing A/B testing	A new competence area for most testers A/B may require good technical platform
	Finding product priorities	User experience testing Product analysis Analysis of usage logs	

Type	Special challenges	Specific solutions	Notes
	Testing of user experience	User experience testing Product analysis	
	Providing a solid version where technical bugs don't affect results	Proper traditional s/w testing (done in lightweight manner – exploratory testing emphasised)	
Fast prototype-based product development	(See much of Fast MVP-based first product development)	Rapid testing of prototypes made with any technology	
Bringing a product to existing, competitive market	Understanding the important criteria in competition, the success factors	Testing of competitors	
	Understanding how the product compares with competitors	Comparison testing, benchmarking Using relative product quality metrics	Comparison of UX and technology
Introducing a disruptive product	Is new concept understood?	Concept testing	Analyse and test in relation to understood concepts
	Is the new concept desirable	UX and other testing	
Agile refocusing of products (pivoting)	Refocusing of testing	Testers with versatile competence	Skills, experience
	Abstract test automation test design technology	Model-based testing Domain-specific languages	Anything that separates abstract actions from implementations
	Good user experience and usability testing	Usability and user experience testers	Use of consultants when making critical decisions is advisable
Fast new product pipeline: individual products	Fast testing of new concepts	Prototype testing Preference testing	Testing of concept is often about UX testing
	Good testing for innovative new features	Fast UX and usability testing and analysis Good exploratory testing & other manual testing	UX testing can be rapid, but should be done properly "Speed requires brains"
		Strict, professional processes	Strictness results for good system design, not from bureaucracy
Fast new-product pipeline: product families	Solid basic test system for platform technology	Good test automation Best platform-specific tools	Need to select tools carefully – but consider expansion to other platforms

Type	Special challenges	Specific solutions	Notes
	Test design at high abstraction level	Keyword-based testing Model-based testing Domain-specific languages	For example Robot framework offers various abstraction levels defined in domain & culture specific means
Fast new-product pipeline: product family for many platforms	Tools that work on many platforms	Platform-independent workflows Robot testing	Platform independence still needs adaptation to each platform, where test robots can help
	Amount of testing for any new version launch (simultaneous for all platforms)	Test automation	Goal is to test all versions in parallel – virtual machines are cheap
	Management of test system architecture	Simple, platform-independent systems OS abstraction	

4.4. Feature development

Table 6. Feature development challenges and corresponding testing solutions

Type	Special challenges	Specific solutions	Notes
Rapid introduction of new features to match competition, needs, changes in environment	Good testing of new features	Good exploratory testing & other manual testing	Testing here need user and business approach – UX emphasised
	Handling of regression	Solid regression test automation	
	Rapid starting of testing of new features	Exploratory testing Automated test generation from designs Automated test generation from implementation	All testing need to start fast

4.5. Reaction to technical changes in ecosystem

Table 7. Rapid reaction to technical changes and corresponding testing solutions

Type	Special challenges	Specific solutions	Notes
Rapid introduction of new or changed functionality to match changes in interoperating technology (API changes etc.)	Testing for changes in external APIs (such as social media APIs)	Automated regression testing against latest and upcoming APIs	Need to monitor what is upcoming from external systems – their roadmaps, development versions, betas
Rapid introduction of new or changed functionality to match changes in new OS versions	Good testing of new functionality	Good exploratory testing & other manual testing	
	Testing for changes in OS APIs	Automated regression testing against the new OS version	Need to monitor what is upcoming from external systems – their roadmaps, development versions, betas
Fast fixes, security and functional updates	Fast testing and deployment, ability for continuous deployment	Testing integrated into continuous deployment workflow	
	Managing regression	Good regression testing	
	Fast testing for large and complex systems	Optimisation of test suites, test runs	Complex systems need complex analysis…. good tools

4.6. Platform changes

Table 8. Rapid change of product platform or ecosystem

Type	Special challenges	Specific solutions	Notes
Change of platform	How to get testers to learn the specifics of new platform	Use portable technologies and testing tools – less to change Hire competent testers who have platform-independent competence of experience from more than one platform Emphasise exploratory testing while automating proceeds	For example Robot Framework in platform independent – just adaptation changes

Type	Special challenges	Specific solutions	Notes
Going multiplatform	How to manage the multiple test assets and testing work	Use high-abstraction level test designs that just need different adaptation Use platform-agnostic testing methods (such as robot-based)	

4.7. World of startups

Table 9. Challenges for startups moving to growth phase

Type	Special challenges	Specific solutions	Notes
More customers or more products	Planning how testing is done	Overall testing process development	Considering all aspects of good testing, including competence, manual and automated testing; all relevant test types
	Building a professional testing systems	Rent environments from cloud	Turn-key systems for startups will be more and more available
	Select test automation that scales	Choose widely used industrial-grade systems that support many platforms (because plans about those will change)	
	Starting from small	Develop automation gradually, incrementally Invest first for good manual testing	Even if building the testing systems is important, it must not hog the focus of the company or even the testing people

4.8. Safety-critical systems

Table 10. Challenges for safety-critical system development

Type	Special challenges	Specific solutions	Notes
General issues	Faster testing speed in general	Use of test automation	Need to be aware of the risks in relying on automation
		Common technologies for safety and functional architectures enables using same tools	
Market entry, certification	Meeting safety standard requirements	Choose a safety standard that does not have excessive testing requirements	Machine builders prefer ISO 13849 rather than IEC 61508. What standards are used needed depends on markets and other factors.
Validation testing (for certification)	Fast validation of the overall system	Separate safety-critical and not safety-critical systems carefully and optimise testing for both	
		Use physical test automation (robots) for system UIs	
Reducing focus for testing after changes	Reduce focus with confidence and proof	Use tools to identify effects of change	Regression testing of everything is time-consuming
Reducing time spent on documentation	Reducing time spent on test documentation	Use product / test / lifecycle management & testing tools that produce interlinked & traced documentation “automatically”	

5. Main principles

The main principles for supporting raised speed with testing are (at least) these:

- Understanding what is done. Make testers understand the goal of development and the customer’s needs so they can prioritise and focus their actions and make compromises elsewhere when speed is critical.
- Don’t go for speed by compromising quality. Learn to do things properly first, then add manageable speed. That means that that only add speed, if your testing can handle it, and your testing can make the increased speed manageable.
- Have a solid testing approach to keep the platform in good shape so direction changes can be made rapidly.
- Use robust test automation that doesn’t break workflows.
- Use intelligent manual testing, because automation never notices everything.
- Testing must be able to show differences to previous release to customer and do comparisons.

- Delivery workflows must be able to be stopped if testing tells that quality is not sufficient.
- Have everything that done and tested under strict configuration and version control.

6. Some final thoughts

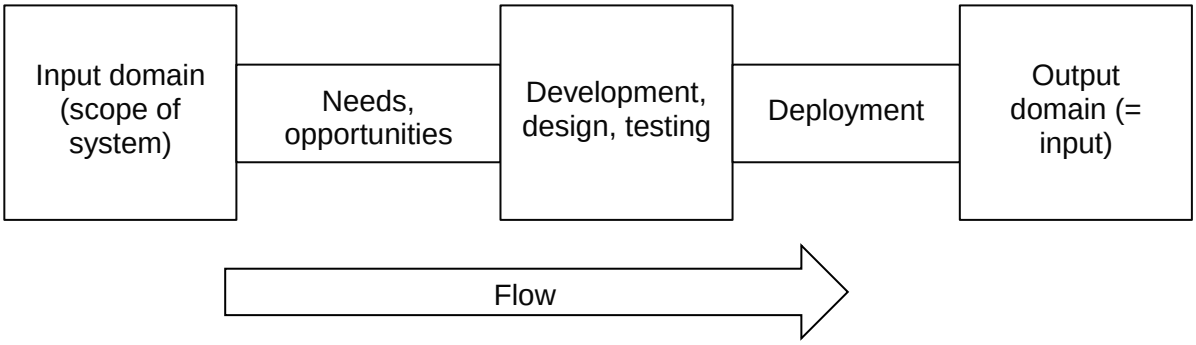
Does speed risk quality? If not managed properly, yes. But sometimes speed produces quality when it keeps the difference between needs and implementations as small as possible. Testing has a central role in maintaining both customer centric and technical quality.

There are many pitfalls in trying to improve speed. It makes sense to differentiate the capability for speed and the actual speed used, which is a matter of business.

To fully support activities and decision making at product development level (and not only at engineering level), tester roles and competences must be – or can be – rethought.

APPENDIX: Product development system as an acoustic system

To understand the dynamism of the product development, let's consider it as an acoustic system consisting of volumes and connecting pipes that have length and diameter. This is because even "continuous" activity is not truly continuous, but information and activity always moves in a rhythm.



A thicker pipe can pass more information, but only if the frequency of the system is suited to the frequency it is driven. The basic ideas in acoustics are that a shorter pipe has a higher frequency. When it is connected to a volume, the smaller the volume is, the higher the frequency and vice versa. And the thicker the pipe, the higher the natural frequency is and vice versa.

Using this analogy we see that we can achieve higher speed if:

- Input domain is restricted. That helps us understand better the requirements of the system and control the focus and size of the developed system. It is also easier to understand risks, design and execute the necessary testing. Smaller, more focused systems have less defects and require less testing than systems that have a wide, non-focused scope.
- There is a short path from customers to development. That way, information can flow fast; there are fewer errors in understanding things. This is why agile teams emphasise direct customer participation.
- The resources for development and testing are large.
- The deployment pipeline is short. Of course, it must not be too short, as it needs to contain the checks that the new functionality is something that should be passed to customer. But many of the checks can be integrated to the development "volume".