DIGITAL INDUSTRIES SOFTWARE

Agile, software requirements management and regulatory compliance: a practical Live approach

Executive summary

Agile methods have proven their ability to improve project success rates, but there is still some pretty wild, yet-to-be explored territory. For example: how can you support information traceability from software requirements elicitation onward while managing risk of noncompliance to industry standards and regulatory mandates using Agile? This paper presents the Live approach and discusses how the adoption and refinement of Agile methods are able to significantly reduce risk of project failure and increase efficiencies of regulatory compliance.

Contents

Introduction

Agile principles evolved to address the perceived limitations of Waterfall development – mainly that Waterfall does not show results until the end, engages stakeholders too late, and unnecessarily delays testing. Agile as a software development approach has gone mainstream, because Agile is focused on keeping the customer happy and gaining a clear understanding of customers' requirements.

The traditional Waterfall model strengths can generally be characterized as plan-driven models of well-defined processes of planning; firm requirements, requirements traceability and testability, and clearly defined acceptance criteria are paramount. The strength of these methodologies lies in the comparability and repeatability that stem from standardized processes. Waterfall development is generally considered to be the least risky development model, which makes it popular for large or long software development projects, particularly in industries with project or product exposure to risk of life, limb, or liberty monitored as such by government bodies.

But the reality is that no organization is a purist following any single prescriptive methodology. Rather, we are "blenders," mixing what we need from Waterfall, Agile (Scrum, eXtreme Programming), Rational Unified Process (RUP), Spiral, or other methodologies into what we need for our projects and organizations to succeed.

To accommodate these blended hybrids, organizations are replacing legacy secular tools that create islands of inefficiencies and exclusion zones with application lifecycle management (ALM) tools that are unified by design, web-based, and easily customizable to support multiple, continuously evolving processes.

This paper describes how some of the most widely adopted best practices, especially the adoption and refinement of Agile methods, have significantly reduced software development risk, in terms of regulatory compliance and increased project success rates. These guidelines are referred to as "Live approach" because they are based on hybrid Agile-Waterfall principles using just-in-time data provided by modern ALM solutions.

It has been proven that to outperform with Agile methods, R&D people must live together in a stimulating environment with few or no distractions relating to progress reporting, discussions with management, document fulfillment and so on.

As an example, consider the approach to gathering eXtreme Programming (XP) requirements ("user stories"). The customer (or user) should be an integral part of the development team, answering developers' questions in real time, rather than an external entity. However, this is rarely achievable in practice because very often the customer is an organization with thousands of employees spread over several dispersed countries, and having complex definition and approval processes for requirements.

Furthermore, Agile teams meet very often to decide what they will achieve in the next few days or even hours. But such best practices can frustrate managers and executives in a very short time. These people need long-term planning and strategic corporate governance of project costs. They need milestones and deliverables, not a day-by-day assessment of "what will we achieve today?"

The Live approach to project information handling can help companies reconcile these disparate but equally vital needs. Three major areas of interest around Agile software development can benefit from the introduction of tools supporting the Live approach: corporate governance, requirements management and project management. Any such new-generation tools and methods must make software requirements engineering, project planning and corporate governance directly involved in software R&D, while keeping the R&D teams Agile and not adding extra work or distractions.

The Live approach is not a methodology like XP, Scrum or RUP. Rather, it is a set of guidelines whose aim is to define a possible roadmap for software development environments and tools to make them open to support different development methods with a higher degree of usability, and able to provide "live" information about project status.

Value propositions of Agile principles

Agile methods have proven their ability to increase project success ratios. The fairly wide adoption of several of them, especially XP, dynamic systems development method (DSDM) and Scrum, proves that most of the principles behind the *Agile Manifesto*[1] are valued by customers and by developers.

For instance, customers love these statements in the *Manifesto*:

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
- "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
- "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."

On the other hand, developers very much like the following:

- "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."
- "Agile processes promote sustainable development."
- "Simplicity – the art of maximizing the amount of work not done – is essential."
- "The best architectures, requirements, and designs emerge from self-organizing teams."

Most common Agile methods

Let's look briefly at some of the most commonly used Agile methods. The most relevant characteristic practices for the discussion that follows are cited in the points below. A broader dissertation can be found in Agile Software Development Ecosystems2.

Dynamic systems development method (DSDM)
DSDM is an outgrowth of, and extension to, rapid application development (RAD) practices. DSDM boasts the best-supported training and documentation of any Agile method. DSDM's nine principles include active user involvement, frequent delivery, team decision making, integrated testing throughout the project lifecycle and reversible changes in development.

Extreme programming (XP)
XP preaches the values of community, simplicity, feedback and courage. Important aspects of XP are its contribution to altering the view of the cost of change and its emphasis on technical excellence through refactoring and test-first development. XP provides a system of dynamic practices, whose integrity as a holistic unit has been proven. Among others there are practices like the daily stand-up meeting and direct involvement of the customer.

Scrum

Scrum provides a project management framework that focuses development into 30-day sprint cycles in which a specified set of backlog features are delivered.
The core practice in Scrum is the use of daily 15-minute team meetings for coordination and integration. Scrum has been in use for nearly ten years and has been used to successfully deliver a wide range of products.

It is clear that the most widely adopted Agile methods completely support key Agile values:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

Staying Agile: is it possible

The aforementioned Agile values, however, are not applicable in every environment. Consider an international corporation with its software R&D spread over three continents, with R&D serving other departments of the organization located all around the world.

Is it possible to locate the "customer" together with the R&D team? What about processes, multilingual manuals, corporate and R&D budgeting, and delivery – plus resource planning?

In big companies, the software development activity (among others) is increasingly being outsourced to offshore premises and providers, creating a growing need in requirements specification and project progress control. In all these situations, code is not the only artifact to be produced.

These facts (and many more) are the foundations of the Capability Maturity Model Integration (CMMI) evolution[3]. The aim of CMM was to certify the development ability of a software R&D team, while CMMI certifies much wider business processes inside an organization, including software development. Another very interesting reason for moving CMM into CMMI is the need for integrating corporate processes and compliance with software development. In light of this, CMMI and Agile methods seem to be incompatible, due to the much broader coverage of the former compared to the latter.

The Agile conundrum: requirements and governance versus development

Over the years, Siemens Digital Industries Software has completed many large-scale implementations where we are contracted to integrate teams of Agile development with the organization's desire for increasing levels of CMMI compliance. "Extreme Programming from a CMM Perspective"[4] also tackled the issue of integrating XP and Scrum with CMM and found that there are some areas in CMM that fall outside the coverage of the considered Agile methods. Such uncovered areas affect process and project control and the monitoring and control of suppliers.

Another interesting perspective is related to customer involvement. For complex systems this often cannot be solved by having the customer sit with the R&D team because formal requirements gathering and refinement is performed by dozens or even hundreds of people, geographically dispersed as often as not. Agile Requirements, Methods and Tools 13.35 states that requirements definition activities nearly always produce documents that are directed from writer to reader, such as from the customer to R&D, and frozen – that is, not supporting change – and this is in direct conflict with iterative and change-driven Agile methods. Furthermore, since Agile methodologies require team collocation and high domain knowledge, if the contractors or outsourced partners are working off-site, it cannot be Agile.

Another interesting fact: "In spite of what many people think, it is not true that Agile methods are without artifacts, although they are certainly less documentation-focused than traditional techniques. Still, this is an issue for organizations for whom the CMMI is the basis for rating their organization."[6]

So software requirements management, as well as risk management, regulatory compliance, corporate governance, and project management disciplines in large or distributed organizations can actually suffer from the introduction of Agile methods in R&D. Is a reasonable compromise even possible?

The answer can be found in one of the principles of the Agile Manifesto itself:

"Give them the environment and support they need, and trust them to get the job done."

This could be interpreted as "give developers a toolset that is fully integrated in the wider processes of the company and let them collaborate remotely as if they were on the same site as their customer, and let their work be seamlessly audited and controlled."

In practice this means that while developers are coding in an Agile world using Agile tools, other people in the company must be able to define and refine requirements, submit changes, manage test cases and track project status using their favorite methods and tools.

Is that realistic? Are tool vendors providing anything that addresses this need?

Software development tools: state of the art

Software development and its supporting environments and tools are entering an historic time. The actual proposition of tools follows a pretty old concept that is strongly rooted in the Waterfall model of software development, with some exceptions.

The "bad old days": disparate tools and data

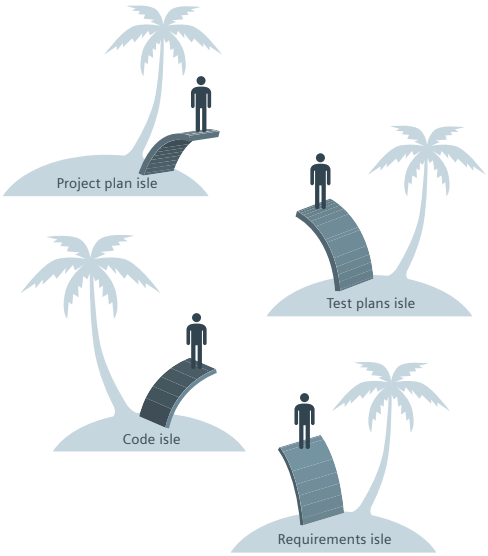
The reason for disparate tools and data must be researched in past achievements in providing support to each phase of the software development process: at some point in time, for example, it was clear that in order to support collaborative programming, the development community needed version management solutions. Some time later, it became evident that software architects and their customers needed new tools for requirements specification and approval in order to manage complex customer-provider relationships. The list goes on and on, covering test management, change request and propagation handling, customer support and other functions.

The problem that quickly arose is that every new toolset that vendors have put on the market defined a new island of automation. Each of these islands, in fact, defined a new data model, a new access policy, a new repository, and so on. Of course it soon became abundantly clear that the information stored in diverse logical data models and repositories had to be integrated. So vendors started building bridges to connect the islands. In many cases, some features that were introduced to support one process were moved into another to extend the support of some information set.

After these improvements, vendors started providing the market with solutions to support versioned requirements, change requests connected to source code, and so on.

Today's state-of-the-art tools proposition

Application lifecycle management is the state-of-the-art proposition in the software development tools market. From the perspective of the aforementioned history of software development tools, ALM represents the latest achievable step on the stairway to integrate islands of automation. Regardless of what analysts may preach, legacy vendors will never achieve ALM with dinosaur code because they cannot economically or ethically ditch laggards paying annual maintenance fees. Legacy vendors respond to the ALM movement by trying to provide their customers with expensive, unreliable one-off integrations between tools proposed by different vendors. This level of effort is nothing more than simple data exchange between tools, anti-Agile, and certainly not the recommended Live approach.



The new breed of ALM

A close look at the new breed of ALM propositions reveals the following common characteristics: ALM solutions aim to be broad, with wide coverage of feature sets and support for multiple project roles and deep, extensive support for the feature set needed by each project role.

ALM solutions nearly always come with interesting guidelines or even full methodologies to support software development. Methodologies and tools are, of course, integrated. Integration (as well as breadth and depth) is what customers expect in ALM solutions.

In addition to integrations, customers expect corporate governance, risk management, compliance management, full project progress tracking and project-related cost control over dispersed teams. Why not? A true ALM system is unified, with consolidated linked data and work items that can be easily search queried and presented in the organization's standard reports without having to rely on developers, IT or contractors.

In conclusion, ALM solutions offer broad support to people covering different roles in software development, deep feature sets for each of them, and integrated functionalities to bridge the information islands created by the different tools comprising the suites.

Integrating Agile with ALM

Gartner information technology research and advisory company agrees that Agile should integrate ALM for best results. In their "Key Issues for ALM," they explain: "Projects deploying Agile methods, geographically distributed projects – in which applications are built and maintained by teams working worldwide, and complex process and product development situations – all benefit from more effective ALM." A hybrid process allows you the flexibility you need with the benefit of greater control.

Even if such ALM solutions have been widely adopted by many companies, they are not much appreciated by Agile teams for several reasons:

- Integrated tools that support their "integrated" methodologies are perceived as not Agile culture friendly.
- During Agile development, all the activities run in parallel. Legacy ALM tools integrations include batch transport of information from one repository to another (bridges between islands), preventing instant notification of changes.
- ALM solutions support different roles with different tools having different processes. Agile processes force teams to live together in the same room, use the same tool and the same method.
- One of the most appreciated payoffs of Agile methods is interchangeability of people. ALM leans more toward project role specialization.

The Live approach can solve the problem of integrating Agile development teams into a wider company/corporate infrastructure by providing developers with live and available access to the wider corporate information via the tools they prefer (and need) to use.

So, to sum up what has been discussed thus far:

1. Using Agile methods in software development gives good results.
2. Insulating Agile teams is pretty difficult in many situations: they are very often part of wider and not-at-all agile corporate processes.
3. Available software development tools are inadequate for Agile teams, which must be involved in wider corporate processes such as risk or
compliance management.

The Live approach

The Live approach from Siemens Digital Industries Software consists in a set of guidelines and taxonomy. Guidelines introduce a new philosophy in managing software development artifacts and development-related information. From these guidelines come a set of criteria defining taxonomy to check the level of adoption of the Live approach in development environments and tools.

Live approach guidelines

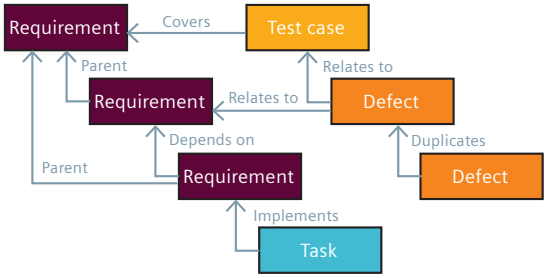
The Live approach guidelines (referred to hereafter as Live guidelines) can be divided into two main areas:

1. Relating to the Live information, the information model and storage (guidelines 1 to 4)
2. Relating to information availability, the way in which the Live information is accessed and exposed (guidelines 5 and 6)

Guideline 1: Single ancestor

The work item class is the common ancestor in an inheritance hierarchy of all the information and artifacts related to the development activities. Its instances are named work items.

More informally, this guideline says that all the artifacts to be created and activities to be performed in the software lifecycle (such as, for example, requirements, change requests, tasks, test plans) are work items. This means that everything managed during software development, from requirements to code, is an instance of work item class.



Guideline 2: Single source

The instances of the work item class and the instances of all its specializations are "single source" – all project information exists only once in the development environment. As an example, consider a test plan that is created during requirements specification. It must be the same test plan connected to the requirements that generated it, and to the defects found during its execution (never a copy of it).

Guideline 3: Single repository

The repository where work items are stored should be logically unique. This does not necessary mean that the repository is physically one, but the repository must appear unique from the user perspective.

While it is theoretically possible to build a logical single repository on top of an integration of multiple repositories, this practice is not recommended because it will probably lead to a situation like the bridge-building between islands of automation. There are some exceptions to be considered; for example, to support scalability needs where multiple repositories provide better performance with mirroring, load balancing, remote replication, and so on.

Guideline 4: Custom work item class specializations

Users can define their own specializations of the work item class to match their corporate or project needs. Examples can be more or less the usual "requirement," "change request," "task," "source file," "code change log," but also "customer purchase order" or whatever makes sense for the organization or just for a single project. This possibility includes the customizability of the information to be stored in the work item class specializations and the work item class itself.

Guideline 5: Live features

A feature is Live when it is an operation applicable to any instance of the work item class and of its specializations. In other words, a feature is Live when it can be applied to any work item.

As a useful example, consider the "show progress" operation that typically applies to a task to get the actual progress of it. Promoting such functionality to become "Live show progress" makes this feature applicable to any work item; so it is possible to get the actual progress on a test plan, on a requirement specification, on a change request, and so on.

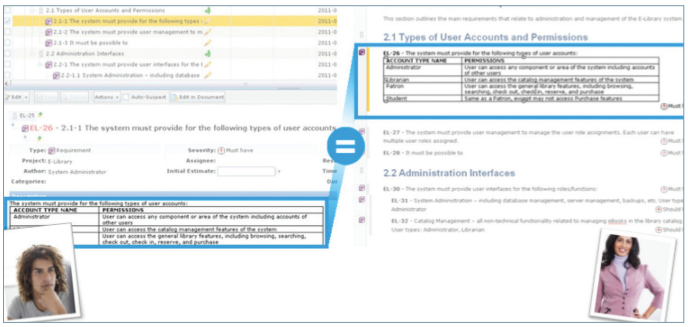
It should already be clear that the larger the number and the greater the power of the Live features in a certain lifecycle management solution, the higher the benefits for its users.

Creating new Live features should represent a stimulating activity for the provider of vertical tools: until now they've been refining features for a certain phase in the software lifecycle and/or for a certain role in the development chain; now they should imagine how these features could be extended to offer their value to every role in every phase of software development.

Consider, as another example, the benefit of having Live impact analysis. This means having the actual requirements-oriented link navigation needed to find the requirements impacted by a change, promoted in a way to wider navigation in order to find every development artifact and activity impacted by the change. So performing a Live impact analysis operation will enable the user to easily find all the activities that were performed to implement a requirement, their cost, the people involved, all the artifacts to be changed (such as source code and user manuals), plus eventually the impact of the implementation of the change on the project plan deliverables at certain milestones.

Guideline 6: Exposure

When using Live features to access work items, the resulting information should be exposed in a way that is appropriate for every single user role. This means that Live features should be available to different user roles in the preferred format and with the specific content desired by the users covering a role.



Same data – different views.

Live levels

This section introduces the Live approach compliancy taxonomy. The taxonomy contains a set of criteria against which to check any development environment (that is, any kind of development infrastructure and access toolset) to state its level of compliancy with the Live approach: such compliancy levels are hereafter referred as Live levels.

The taxonomy contains five Live levels. The last level provides full compliancy to all the guidelines, plus a rich set of Live features correctly exposed to each different user role. Although the last level should stand as the final state to be reached by every toolset at the end of the development tools revolution, intermediate levels 1 to 4 are defined as a map for getting there. These intermediate levels should help companies to assess the actual degree of support for the Live approach by their development infrastructure, and suggest further steps for improvement to move to the next level.

The five Live levels are:

- Level 1 - Foundation
- Level 2 - Connection

Govern	**Live dashboard**
Control	**Live plan**
Fusion	**Live track**
Connection	**Live trace**
Foundation	**Live search**

- Level 3 - Fusion
- Level 4 - Control
- Level 5 - Govern

Each level contains criteria to evaluate the compliancy of the software development environment against Live guidelines. Each level extends the criteria of the previous level. In what follows, the criteria are discussed separating the Live guidelines according to their area: Live information and information availability.

For example, project managers will want access to project progress information in a plan format, while executive managers will want to see only critical paths in reaching milestones summarized in a dashboard, while developers will see the tasks assigned to them and their deadlines directly in their integrated development environment (IDE).

It is clear that each development environment can be compliant to these guidelines at a different level at a certain point in time.

Live information

This section introduces the criteria to evaluate the level of compliancy of the software development environment against Live guidelines 1 to 4. So, these criteria are related to the way in which the information is organized: data model and storage.

1. Foundation level. At this level, guideline 1 (single ancestor) and guideline 2 (single source) must be supported. So the work item class is defined as common ancestor and all its instances and its successors' instances are single source.
2. Connection level. Work items are connected through links. Links must support different connection types according to link roles. So work items can be connected with "containment" links or "impact" links, for example.
3. Fusion level. At this level, guideline 3 (single repository) is supported: work items are stored in a single logical repository. Additionally, the repository must support version and history management on work items and links, plus work item workflow management with statuses, transitions, assignees and user notification mechanisms over at least status changes. Finally, the repository must guarantee a secure access.
4. Control level. Work items store information related to time, priority and cost, support discussion and approvals. Note that the content of time, cost and priority information is broad: these may include estimated time to completion, planned start, planned end, assigned project milestone, expected cost, actual cost, value, added value, priority, severity, etc.
5. Govern level. At the last level, guideline 4 (custom work item class specializations) is supported. Link types are user-defined as well. So work item types, content and connections can be customized based on user, project and corporate needs. Risk, resource and financial management related information are added as well.

Information availability

This section introduces the criteria to evaluate the level of compliancy of a software development environment against guidelines 5 and 6. These criteria are related to the way in which the information is managed: features and their exposure.

The Live features to be added at each level are:

1. Foundation level. Live search: work items are searchable by means of every attribute.
2. Connection level. Live trace: work items support link role-based navigation, and impact and traceability analysis.
3. Fusion level. Live track: extended lifecycle management for the work items.
4. Control level. Live plan: project planning and progress where all the work items, or work items belonging to selected specializations of the work item class, appear in a self-updating plan. This means that the plan is automatically created from the information stored in work items (such as priorities, severities and dependencies) and recreated as a result of any change to planned work items.
5. Govern level. Live dashboard: to govern every project activity in real time. The dashboard is also able to show multi-project information such as resource workload and cross-project code re-use, for example.

At every level, the Live features must provide information to the user in the appropriate format for the user's role. In the "island of automation" approach, there is only role-specific information available in the appropriate format for users covering a role. In the Live approach, all the information is available to all users in their desired format.

As an example, consider project leaders, who deal with project plans and Gantt charts. To get a view over the status of their projects, they must stroll around looking at requirements contained in Office documents, issues contained in trackers, code stored in versioning systems, and so on. With the Live approach, the actual status of every work item (that is, the status of their project), is directly available for them in a plan format.

The information provided by Live features at every level must be available to every user. Thus, if the results of a Live search included in Live level 1 can be provided to every user by means of a unique web interface, at Live level 4, different users will deal with the Live plan in a different way: developers reporting their progress in their IDEs, project leaders arranging the priorities on a Gantt, managers looking at money consumption in a spreadsheet where they can re-assign budgets.

Level	Live information	Information availability
Level 1 – Foundation	The work item class is defined as common ancestor and all its instances and its successors' instances are single source	Live search
Level 2 - Connection	Work items are connected through typed links	Live trace
Level 3 - Fusion	Work items are contained in the same versioned environment supporting change and workflow management	Live track
Level 4 - Control	Work items store time and cost related information	Live plan
Level 5 - Govern	Work item types, content and connections can be customized	Live dashboard

Table summarizes the criteria for Live levels compliance.

The Live approach and Agile development

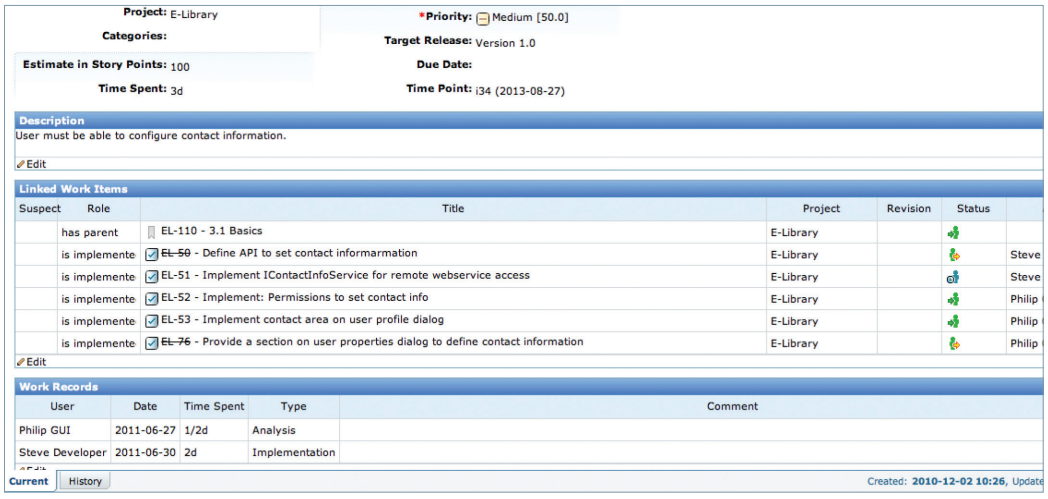
The Live approach can be applied easily to bridge the gap between Agile (in the R&D team) and formal (outside R&D) processes using tools that provide Live and available project information starting from Live level 4, as specified in the previous section.

At Live level 4, for example, requirements, tasks, project milestones and project cost information as well as change requests, test plans, features, source code, builds, etc. all exist "single-source" in a versioned, fully traceable, and workflow-driven repository. Additionally, all relevant information for each corporate or project role is available in the role's preferred format.

Example: Complex requirements inception

As an example, given appropriate Live approach tools, a complex requirements inception phase with refinement and several approval levels can be performed inside the client corporation in Atlanta by means of Office documents in the same environment where the project manager in Munich specifies tasks, priorities and milestones in Gantt format, and where the development team in Delhi tracks their artifacts and progress in an Agile way.

When moving to Live level 5, such dispersed teams will be seamlessly providing the executive management in San Francisco with crisp information that reveals delays, bottlenecks, costs and risks.



Conclusion

Agile methods, representing a kind of rejection of all the infrastructures of methods and tools built in the last decades to produce software, have defined a new, successful, and "free" way of creating working code. Unfortunately, Agile methods are not always practical to apply due to risk management, compliance management or process-oriented environments of larger and more dispersed companies.

Staying agile in software R&D departments and still matching corporate needs is possible thanks to a new category of software development tools that has in fact already emerged in the market and is rapidly making significant inroads in companies that have experienced the dilemma identified in this discussion: the need to apply proven Agile software development methods within a wider, less Agile (or non-Agile) corporate context. An example of such a tool is Polarion® ALM from Siemens Digital Industries Software, which actually stands at level 5 in the Live levels taxonomy.

The ability to mix the benefits of Agile software development and more formal requirements management, planning and governance methods of the Live approach opens new directions for future research in creating vertical Live methodologies and tools to support the needs of different business sectors.

References

1. *Manifesto for Agile Software Development*, http://agilemanifesto.org
2. J. Highsmith, *Agile Software Development Ecosystems*, Pearson Education, 2002
3. CMMI, http://www.sei.cmu.edu/cmmi/
4. M. Paulk, "Extreme Programming from a CMM Perspective," *IEEE Software* 18.6, 2001
5. R. Davies, *Agile Requirements, Methods and Tools* 13.3, 2005
6. D. Kane, S. Ornburn, "Agile Development: Weed or Wildflower?" *InformIT*, 31 Aug. 2002

Polarion® ALM™
The unified application lifecycle management solution.

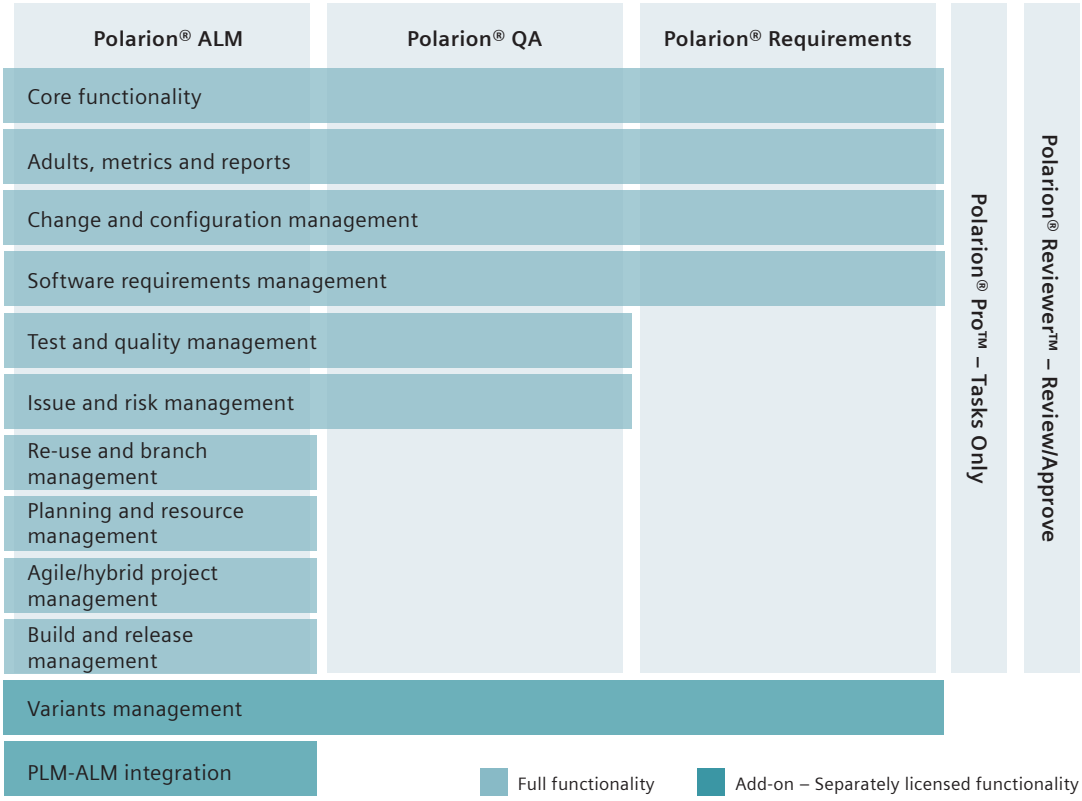
Connect teams and projects, and improve application development processes with a single, unified solution for requirements, coding, testing, and release.

Polarion® Requirements™
Complete software requirements management solution.

Effectively gather, author, approve and manage software requirements for complex systems across the full project lifecycles.

Polarion® QA™
Complete test and quality management solution.

Design, coordinate, and track all your test management activities in a single, collaborative QA environment.



Siemens Digital Industries Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. Our solutions help companies of all sizes create and leverage digital twins that provide organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit siemens.com/software or follow us on LinkedIn, Twitter, Facebook and Instagram. Siemens Digital Industries Software – Where today meets tomorrow.

About the author

Dr. Stefano Rizzo is a product leader and visionary at Siemens Digital Industries Software. He has some 18 years of IT consulting and mentoring experience in several different business areas including finance, telecom, software and government. As a mentor he has helped dozens of big companies introduce new development processes and methods. As a teacher he has trained thousands of people in UML, requirements management, and Agile development. As a methods evangelist he has helped hundreds of companies to share his vision about collaborative software development. His actual focus now is researching and developing methods and best practices for Agile and Live software development processes.

siemens.com/software

00000-D1 6/21 C