A Systematic Comparison of Agile Principles and the Fundaments of Component-Based Software Development

Iva Krasteva
Faculty of Mathematics and Informatics, Sofia University, 5 James Bourchier Blvd, Sofia 1165, Bulgaria
Iva.krasteva@rila.bg

Per Branger, Rikard Land
Department of Computer Science and Electronics, Mälardalen University, Västerås, Sweden
per.branger@mdh.se, rikard.land@mdh.se

MRTC report ISSN ISSN 1404-3041 ISRN MDH-MRTC-220/2007-1-SE,
Mälardalen Real-Time Research Centre, Mälardalen University, December 2007

Table of Contents

Introduction

This report presents a systematic comparison of the principles of agile software development and the fundaments of component-based software development with COTS (Commercial Off-the-Shelf) components. The fundamental assumptions and inherent characteristics of the two fields are compared, and any theoretical incompatibilities are reported. The study is limited to include only development activities, which are [31]: requirements, design, development, verification and validation, and integration. We do not consider activities such as project management, configuration management, maintenance and evolution, and documentation. Furthermore, the study concerns development with COTS components, not other types of component-based development, such as [15]: product-line development (where components are built in-house) or architecture-driven development (i.e. top-down design decomposition resulting in components to be developed in-house).

This theoretical study should be seen as a first phase, laying the foundation for further empirical studies in an industrial setting. These two steps are well-defined parts of the research agenda of the established PROGRESS Centre for Predictable Embedded Software Systems[1] and also the ITEA2 FLEXI project[2] of which we are part.

[1] http://www.mrtc.mdh.se/progress/
[2] http://flexi-itea2.org/index.html

Research Method

When comparing two independently evolved paradigms, one first step is to bridge all gaps between differences in their respective self-representation, i.e. terminology and concept formation. During this type of comparison, we also need to identify (or construct) the fundamental set of "facts" of each field so that the majority of practitioners and researchers in this field would agree with this choice, and make this choice explicit. At each step of the logical reasoning, we have been careful to document the basis for our conclusions, to make choices explicit and to motivate them, thus opening up our work for external scrutiny and criticism.

The research follows the following structure: For each of the listed development activities (requirements, design, development, verification and validation, and integration [31]), we:

1. List the 12 agile principles according to the Agile Manifesto [8] and describe how they apply to that development activity. Although a (subjective) step of interpretation and application is needed, we have made sure to externalize the interpretation of the principles as much as possible by supporting our conclusions with practices of different agile methods- mainly XP [5][6] [7], Scrum [30], and Crystal Clear [11].
   Some of the principles relates to the whole development process or supportive process activities such as Project Management and are excluded from the discussions.
2. Describe what makes this activity special for:
   - The development of systems with COTS components (compared to system development without COTS), and
   - The development of COTS components (compared to development of any product).
3. Identify conflicts, and outline possible solutions, between:
   - The agile principles and the development of systems with COTS components, and
   - The agile principles and the development of COTS components.

   In addition, we note any comments believed to be relevant to consider and give a second thought when applying the agile principles to COTS-based development. The observations that we make are summarized in suggestions for application of agile ideas in particular development activity.

The rest of the report follows a structure where each activity has a heading, under which we first treat items 1 and 2 above in separate tables, followed by a section each on system development with COTS components and component development treating item 3.

Requirements

Table1: Applying Agile Principles to Requirements Specification Process Activity

Agile Principles		Requirements Specification
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	P1	Delivery of those items that have greatest value for the customer [12]. In a number of agile methods the principle is supported by practices that say that requirements for a release are prioritized by business value: XP: Planning Game- Release planning [7] Scrum: Sprint Backlog
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	P2	The way to manage changing requirements during the requirements phase is to specify the set high-level requirements at the beginning of the project, which defines the scope and the vision of the project. Details behind each of the high-level requirements can change during the project and they are identified as late as possible. During each delivery cycle some of these high-level requirements are implemented. The details of the requirements are specified no earlier that the beginning of each delivery cycle. Existing practices that support these principles are: Scrum: Product backlog, sprint backlog, sprint planning meeting XP: Planning game
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	P3	*Relevant to the whole process.*
Business people and developers must work together daily throughout the project.	P4	"Onsite business expertise" [12] should be available. Information to and from the business should be easily accessible. In Scrum there is a Product Owner that is constantly collaborating with the team [30]. In XP Whole team practice supports that idea. In XP there is a customer on-site.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	P5	*People, not process aspect /Project Management consideration.*
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	P6	The “default" [29] mode of communication is face-to-face conversation. The specification documents are kept simple and informal. In XP requirements are described by means of stories. In Scrum Product Backlog serves as a requirements specification document. In XP requirements are continuously gathered, clarified and (re)negotiated with a customer on-site.
Working software is the primary measure of progress.	P7	*Relevant to the whole process.*

Agile Principles		Requirements Specification
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	P8	*People, not process aspect /Project Management consideration.*
Continuous attention to technical excellence and good design enhances agility.	P9	N/A
Simplicity-the art of maximizing the amount of work not done-is essential.	P10	Do not make predictions [29] about the future. In Lean development this principle is supported by "Eliminate waste" principle, which in this context is about skipping the unclear and uncertain requirements.
The best architectures, requirements, and designs emerge from self-organizing teams.	P11	*People, not process aspect /Project Management consideration.*
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	P12	*Relevant to the whole process.*

Table2: Requirements Specification Process Activity Characteristics for Agile SD, System Development with COTS Components and Component Development

Agile Process Activities Characteristics	System Development with COTS Components Process Activities Characteristics	Component Development Process Activities Characteristics
Business collaboration and involvement in the process. Requirements are prioritized by business value. High-level initial requirements specification in the beginning and further refinement during each development cycle. Informal, simple requirements specification. Only part of the requirement set is implemented in one development cycle. Unclear requirements are skipped for the next iterations. Prototyping (Spikes (XP), Walking Skeleton(Crystal))	The process of requirements engineering are combined with component selection and the evaluation process [14][20]. Requirements are refined during the selection process [3][9][10][20][26], although sometimes a complete set of requirements is assumed [22]. Requirements should cover the whole system, but not be specified in detail initially, to include many potential components [31]. System requirements can not easily be translated to component requirements, and the requirements definition activity may be closely intertwined with component selection [3][9][10][13][19][20][24][26]. Requirements are not only functional and non-functional, but also architectural and (when selecting components) business considerations [20] (see also [3][9][10][19][24][26]).	Requirements are unclear, inconsistent and even unknown[14]. Requirements accumulate fast in the early stages of the component's life-cycle [14]. Dependability issues should be considered Challenges (necessary but no immediate benefit in terms of #features): backward/forward compatibility, compliance with standards.

Comment: Requirement specification phase in the development process of component-based systems is quite interlaced with component selection process and high-level design specification. That is why in the considerations below, the component selection process will be discussed in parallel and as part of the requirements specification.

Contradiction: The biggest contradiction that exists is about the responsiveness to change and the possibility to introduce change late in the development process of component-based systems. Requirements for systems based on components should be pretty well defined in advance. The reason for this is that changing a COTS component is a very hard task. COTS are delivered to the team as a black-box, sometimes without source code and often without detailed specification. The way of changing a component (if possible at all) is to contact the supplier. This includes sending a mail with the proposed changes, waiting for a response, perform meetings with the supplier, negotiating schedule and costs, etc., which can significantly disturb the development process. That is why introducing changes in the requirements of a component-based systems involves either reconfiguration of components or replacing components. However, both activities are limited to the extent they can meet changes in the requirements.

By component reconfiguration only a narrow set of requirements changes can be satisfied. Component replacement is also not a trivial task. During the component selection process architectural requirements such as component models should be considered. Furthermore, components are evaluated in composition with other components. Another hindrance is that component itself is a set of functions and quality properties and there are no two components that cover exactly the same set of functionality and behavior.

Solution: Although a significant part of the overall requirements specification should be done in advance, the processes of requirements elicitation and component selection in component-based systems are very liable to applying agile principles. Initially requirements should not be specified in too much detail, because it is practically impossible to find a component which fulfils all requirements. Instead, the requirements are refined in more detail iteratively during component selection and evaluation. Through "gap analysis" [20][27] along with the customer the component which gives the most and leaves the least (in terms of effort/cost) is identified.

Consideration: Requirements are not only functional and non-functional, but also architectural and business oriented. Architectural requirements are important in order to prevent from selecting components that satisfy the requirements and have high quality but are incompatible. Business considerations include available component support, vendor reputation and stability etc.

Application: Agile ideas benefit requirements engineering activity for component-based systems in two directions, which support and complement each other:

- making the customer part of the team
- introducing an iterative requirements elicitation and component selection process

Iterative requirement specification is important for receiving an early feedback from the customer. Furthermore, during the selection process it is often needed a trade-off between the required functionality and selected component to be made. It is the customer who has the position to choose one or another of the alternatives according to business value.

Application: During the requirements specification process for systems based on COTS, the architectural and business requirements should be considered along with the functional and non-functional ones [20] (see also [3][9][10][19][24][26]).

Application: An approach that adds additional value in requirements specification of component-based system is prototyping. Prototyping is a common practice for identification and clarification of customer requirements. Prototypes of systems that are based on components are easily produced. However, some policy issues, such as trial versions, should be considered.

Component Development

Comment: The difference between requirements specification of components and particular systems is that components should meet the requirements of many different customers. Initially, "requirements are unclear, inconsistent and even unknown" so the process involves additional step of requirements identification at business level which will be useful in as many different business scenarios as possible.

Contradiction: In component development there are many general users and there can be no specific customer.
Solution: Customer representatives can be used. They can be some marketing people, domain experts or one or more real customers/ users.

Consideration: As component interfaces can't be changed very often issues such as backward/forward compatibility) and compliance with standards, should be decided early in the requirements phase. The requirement on backward/forward compatibility means that "enough" time should be spent early to predict future changes, in order to make those changes easier and backwards compatibility easier - which is in contradiction to agile principles. (Examples: file formats, APIs.)

Consideration: In component development more attention should be paid on non-functional requirements. This often involves technical experts to identify such requirements as they are not always visible by business people or is based on prediction of the way component will be used.

Application: In order to benefit from close collaboration with the business, as agile approaches suggest, some additional steps for identifying a ‘proxy’ customer of a component should be done. Requirements are identified in enough details so that component interfaces are not changed during subsequent releases of a component. Non-functional requirements for a component are specified and addressed along with the functional ones.

Design

Table 3: Applying Agile Principles to Design Process Activity

Agile Principles		Design
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	P1	N/A
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	P2	Late-changing requirements are supported by continual attention to architecture [12]. Refactoring is one of the most common techniques to support the changing architecture and keeping it the best and the simplest for the present moment.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	P3	*Relevant to the whole process.*
Business people and developers must work together daily throughout the project.	P4	A shared vision of the system architecture is created which is understandable by developers and by the customers as well [5]. This principle is supported by Metaphor practice in XP.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	P5	*People, not process aspect /Project Management consideration.*
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	P6	The formal written design is not demanded by agile teams [29]. In XP the Metaphor, which is a common notion of the system design, is a representative of system architecture.
Working software is the primary measure of progress.	P7	*Relevant to the whole process.*
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	P8	*People, not process aspect /Project Management consideration.*
Continuous attention to technical excellence and good design enhances agility.	P9	The good design should be produced in the beginning and be continuously reviewed and improved during the project [12]. The continuous attention to good design is done by Refactoring practice.
Simplicity-the art of maximizing the amount of work not done-is essential.	P10	The simple design is considered those design that is just enough for today's code [5]. It has the fewest possible classes and methods and contains no duplications [6]. In XP Simple Design practice supports these ideas.

Agile Principles		Design
The best architectures, requirements, and designs emerge from self-organizing teams.	P11	*People, not process aspect /Project Management consideration.*
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	P12	*Relevant to the whole process.*

Table 4: Design Process Activity Characteristics for Agile SD, System Development with COTS Components and Component development

Agile Process Activity Characteristics	System Development with COTS Components Process Activity Characteristics	Component Development Process Activity Characteristics
The developers and the customer share common notion of the design Attention on good design is paid at the beginning of and during the whole project System architecture is informal based more on conversations than on documents The simplest design for today's code is produced	The architecture is determined by the requirements, the component model (in a broad sense) and by the component selection procedure [14][20]. An important goal for the design is to minimize architectural mismatch [14], which can be done by considering compatible sets of components as candidates [9][20][25]. There is less need for low-level design activities (since much is embedded in black-box components); instead there is a higher focus on interfaces between components and the mediating 'glue code'. Defining a robust (few changes over time) architecture is critical for successful reuse [31]	Components are more general designed to allow for reuse in various contexts [31], which however increases their size and complexity [14]. Component design should be concrete and simple enough to be efficient [14]. Integrateability is a very important feature of a component and may include documentation of its interface, and the extent to which it supports applicable standards (and documentation thereof).

System Development with COTS Components

Comment: The agile and system development with components design support and complement each other. The architecture specified by means of components, which are "providers of business services at a higher level of abstraction and granularity than traditional objects" [32] is enough simple and understandable for all involved project stakeholders. Furthermore the component architecture helps to decrease the complexity as the system is decomposed on smaller units of business functionality. Both development approaches pay considerable attention on good system architecture.

Comment: The architecture is determined by the requirements, the component model (in a broad sense) and by the components selected [14]. An important goal for the design is to minimize architectural mismatch [14], which can be done by considering compatible sets of components as candidates [9][20][25]. The design activity is focused on interfaces between components and the mediating 'glue code'

Consideration: Similar consideration as for requirements, i.e. replacement of components comes with a high cost in terms of required redesign and reimplementation. As a consequence, developers must predict "enough" of future changes to select "future-proof" components, which is in contradiction with agile principles and practices. (There is a possible exception: if only a standardized interface is used, replacement will be easier; an example is to replace a database component if you use only the standardized and commonly supported part of SQL.)

Application: The design process in system development with components is somewhat dynamic and exploratory, with much feedback between architectural design and component evaluation/selection [20]. This suggests for highly iterative and incremental design process bound with component evaluation process

Component Development

Contradiction: The biggest contradiction between component design and agile design activities is about the simplicity and generality. In component design additional decisions should be made and considered during design specification. One such decision (1) is about component interfaces- how to specify and design the interfaces that provide functionality to be as efficient for reuse as possible, how to make the component as independent as possible by minimizing the number of interfaces that requires functionality, to add configuration interfaces to support the adaptability. Furthermore, the component interfaces should change as little as possible from one version to another so they are usually specified in the very first versions of the component. Extending the interface without breaking the old one is acceptable from the point of view of existing component users, but is practical only to some extent – after a while the interface will look awkward if it is continuously extended without anything being removed. However, these constraints do not apply to the underlying implementation. Another thing (2) that should be considered during the component design is the component technology and supported standards. You probably have to support some standard(s) (including de facto standards and "what everyone expects nowadays") to get any customers at all, but it cost much to implement and maintain support for (evolving) standards and component technologies.
No solution: The overall component design should be specified in advance, so that all the interfaces are kept the same during subsequent versions of the component. Furthermore, additional considerations about reusability should be done when designing a component.

Contradiction: Complexity and additional non-functional requirements for COTS involve a more formal approach to design and architecture than agile methods suggest. However, semi-formal approach for design specification of COTS exist [28]. In order to support component integrateability, a part of the design specification (namely the description of the component's interface) has to be provided with the component so that the component user knows how to use it. However, this documentation may in fact be written (or at least completed) in late stages (after testing), so it is not necessarily part of the design activity.

Application: Involving business people in design activity for COTS-based systems is easily achievable as the architecture specified by means of components, is enough simple and understandable for all involved project stakeholders [32]. An important goal for the design is to minimize architectural mismatch [14], which can be done by considering compatible sets of components as candidates. Similar to requirements, replacement of components comes with a high cost in terms of required redesign and

reimplementation. So developers must predict enough of future changes to select future-proof components.

Development

Table 5: Applying Agile Principles to Development Process Activity

Agile Principles		Implementation
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	P1	N/A
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	P2	N/A
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	P3	*Relevant to the whole process.*
Business people and developers must work together daily throughout the project.	P4	N/A
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	P5	*People, not process aspect /Project Management consideration.*
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	P6	Communication and collaboration is involved in the development process as a means for exchanging knowledge. XP practices, e.g. Pair Programming and Collective Code Ownership support the idea. Side-by-side Programming practice introduced by Crystal is an alternative to Pair Programming and also suggests for intense communication between developers in a project.
Working software is the primary measure of progress.	P7	*Relevant to the whole process.*
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	P8	*People, not process aspect /Project Management consideration.*
Continuous attention to technical excellence and good design enhances agility.	P9	Code is kept good and clean throughout the project. A common practice used in agile methods is Coding Standards.

Agile Principles		Implementation
Simplicity-the art of maximizing the amount of work not done-is essential.	P10	Code should be as simple as possible and just fulfill what is needed for the moment. In XP implementation is connected to producing the simplest code that makes the tests pass. [4]
The best architectures, requirements, and designs emerge from self-organizing teams.	P11	*People, not process aspect /Project Management consideration.*
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	P12	*Relevant to the whole process.*

Table 6: Development Process Activity Characteristics for Agile SD, System Development with COTS Components and Component development

Agile Process Activity Characteristics	System Development with COTS Components Process Activity Characteristics	Component Development Process Activity Characteristics
Code is kept clean and simple throughout the project. Test-driven development is an important aspect of the implementation activity in agile methods.	To a large extent reduced to the creation of 'glue code' and component integration and adaptation [14], but also complemented with more complex implementation of unique system features (however utilizing the components). Component integration is part of the component selection and evaluation process because some of the effects of using a component can be only discovered when components are integrated; compatibility cannot be evaluated in isolation but is always a property on the relation between (at least) two components [9][14][20][25]. Integration could at least in principle be a part of system maintenance because some components could be integrated dynamically during run-time [14]. The ability to integrate a component (its "integrateability") is a very important feature of a component, and must be evaluated during component selection, by e.g. considering how well its interface is documented and to what extent it supports applicable standards.	Nothing specific to the component-based paradigm.

System Development with COTS Components

Comment: The coding activity involves adapting components and writing wrappers and 'glue code', thus building component assemblies to provide system functionality. The development time of system with components is reduced by almost 50% [14]. It involves creation of 'glue code' and component adaptation [14]. Although the time for development is shortened the effort for 'glue code' creation is about three times the effort per line of application's code.

Comment: The integration process is central when a system is developed out of components [21]. We should distinguish the general system integration process from component integration process specific

for component-based systems. Because in component-based systems the parts that are integrated within the system are mostly components we will study the component integration process in this section.

Comment: The integration process is central when a system is developed out of components [21] and should be an integral part of the component selection process. The largest part of system implementation involves adapting components and writing wrappers and 'glue code', thus gluing components together (building component assemblies) to provide system functionality. It is possible that components need to be reconfigured when a new component is added to the integrated system. Issues with component integration exist even in run-time, when components are added dynamically to the system.

Consideration: Since some of component properties become obvious after component integration, test suites created for a component should be updated and/or enlarged with new tests after component integration. (See also our discussion on Test-Driven Development in section High-level Considerations)

Consideration: The ability to integrate a component (its "integrateability") is a very important feature of a component, and must be evaluated during component selection, by e.g. considering how well its interface is documented and to what extent it supports applicable standards.

Application: Agile principles for early and continuous delivery of working software are supported to a great extent of the characteristics of implementation activity of COTS-based systems. Part of component integration is performed during component selection process. The ability to integrate a component (its "integrateability") should be evaluated during component selection Reduced coding time allows for receiving timely feedback. Such implementation process supported by automated tests can be very beneficial and problems to be discovered early and reconfiguration to be done.

Component Development

Comment: No particular constraints and limitations exist.

Consideration: There could be some prohibitive overhead involved in each release which makes too frequent releases impractical, such as the printing of boxes and manuals, component certification [1]. However it must be remembered that an iteration is not the same as a release, it is entirely possible to have frequent iterations internally.

Verification and Validation

In Verification and Validation, we also include testing, which is an important type of verification in agile development. (See also the discussion on test-driven development in the section "Test-Driven Development and Component Selection" on page 27.)

Table 7: Applying Agile Principles to Verification and Validation Process Activity

Agile Principles		Verification and Validation (including testing)
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	P1	
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	P2	The preference towards test automation in most agile methods support to the highest extent the possibility for introducing changes at a later stage.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	P3	Constant testing reduces delivery time [18]. Verification of requirements (have we built it right) can be performed using the product backlog after each sprint, i.e. to see that the software after an iteration fulfills the intended scope in the beginning of the iteration. To demonstrate the application to the customer at the end of the iteration can be considered as validation (have we built the right thing), i.e. is the software fulfilling the customer needs.
Business people and developers must work together daily throughout the project.	P4	Acceptance testing is the area where business people are usually involved in software validation. In XP developers help customers to specify automated acceptance tests for the system [5].
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	P5	*People, not process aspect /Project Management consideration.*
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	P6	The formality of verification and test documentation can be relaxed. Validation can be performed face-to-face in a demo together with the customer.
Working software is the primary measure of progress.	P7	*Relevant to the whole process*
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	P8	*People, not process aspect /Project Management consideration.*

Agile Principles		Verification and Validation (including testing)
Continuous attention to technical excellence and good design enhances agility.	P9	Quality assurance is integrated throughout the lifecycle of most of agile methods [11][16]. In test-driven development, which has been introduced in XP, testing drives the design and development activities. Unit tests are written before the code so the implementation is driven by tests. Tests also help since design decisions are considered and further refined [12]. In most of the agile methods reviews or inspections are held as a mean for sharing knowledge and enhance quality. Pair Programming in XP involves continuous review of code during the whole implementation cycle. "Pair programming can be viewed as the equivalent of instantaneous code inspections. Code inspections are a key part of FDD (Feature Driven Development) and are advocated by other Agile approaches." [18]
Simplicity-the art of maximizing the amount of work not done-is essential.	P10	Testing is done only as much as required by the customer. Usually functional requirements are covered by functional tests written by the customer (and developer). Any particular non-functional requirements that may exist are captured as requirements. The test-driven development approach supports simplicity since it implies that the code to be written should be just enough to make the test pass.
The best architectures, requirements, and designs emerge from self-organizing teams.	P11	*People, not process aspect /Project Management consideration.*
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	P12	*Relevant to the whole process.*

Table 8: Verification and Validation Process Activity Characteristics for Agile SD, System Development with COTS Components and Component development

Agile Process Activity Characteristics	System Development with COTS Components Process Activity Characteristics	Component Development Process Activity Characteristics
Tests are automated and provide test coverage to a highest possible extent. Both unit and functional tests are commonly used. Other types of testing are used if needed Testing is integrated and done constantly throughout software lifecycle. Code inspections and reviews are still held Lightweight test documentation. The test-driven development approach is used (or suggested) in many of methods.	When building a system from components, it is possible to distinguish two separate processes of verification and validation [14]. Verification and validation of the system, with respect to system requirements. Verification and validation of the constituent components (to be selected). You are restricted to black-box testing. Evaluation and testing of components is an essential activity done during the component selection process. [2][20]. Once a component is selected and used in a system, the tests would be stored and re-executed for new versions of the component, to verify that no (bad) changes has been made (at least you discover the changes).	Component development involves more formal verification process to assure good component quality. Bug fixing in maintenance phase is not trivial as far as components are concerned. Component certification by an independent third party is one possibility outlined for the future [1]. There are certain fundamental limitations of component evaluation without a system context [2][17]. One important part of component verification (in absence of a system context) is verification of its "integrateability", i.e. the accompanying documentation of interfaces, its standard compliance and related documentation, and supplied code and applications which illustrate the possibilities of the component (while also teaching how to use it).

System Development with COTS Components

When building a system from components, it is possible to distinguish two separate processes of verification and validation [14] **:**

- Verification and validation of the system, with respect to system requirements
- Verification and validation of the constituent components (to be selected)

System validation and verification

Comment: There are no constraints particular to the usage of components.

Comment: The combination of components in a system needs to be verified and validated. Even if the verification and validation is performed in every iteration, the final verification and validation (acceptance test) before shipment can reveal gaps in expectations. It is sometimes not possible to verify non-functional requirements (e.g. performance) until all components have been integrated in the system.

Component validation and verification

Comment: No contradictions between the agile principles and component validation and verification have been identified. One limitation is that verification and validation is restricted to observations of the external behavior of components, i.e. you cannot apply white-box testing techniques.

Consideration: Since component behavior is known by its specification which is not always sufficiently detailed, comprehensive test coverage is not possible for an acquired component. The

system developer should focus its test suites on the component features desired and/or used in a system, and in practice extensive test coverage will be achieved only for these features. Since components are constructed to be general and to suit different situations and environments, there will be many features which are thus only partly tested by the system developer.

Consideration: Experimentation, formal testing and prototyping would be an excellent way to learn about the component behavior (especially for non-functional properties), and the tests would then be stored to verify that no (bad) changes has been made when a new version of the component is released (or at least it becomes apparent what the changes are, and you have the choice to adapt your system and use the new version anyway). In this way, the automated tests of component features would be used and re-executed many times during the whole system development process: first created and executed during the selection process, then as part of integration testing and system testing, and then during subsequent iterations (if any) as regression testing.

Application (of agile ideas during the validation and verification of systems based on COTS components): When creating systems made of COTS components component validation and verification process is executed early in the development during the selection process. In most cases, only black-box testing of the desired functionality is performed. Test automation can be beneficial not only to assure quality during subsequent iterations of component selection and system development but also to test emergent properties on system integration. Final system testing of the whole system is needed as not all requirements can be verified before components are integrated.

Component Development

There seem to be no particular contradictions; however some specific aspects of component development should be considered, as discussed in the following.

Consideration: An additional complexity is the verification of the component in the absence of a context [2][17], which is fundamental for the idea of certification of components by independent third parties [1].

Consideration: One important part of component verification (in absence of a system context) is verification of its "integrateability", i.e. the accompanying documentation of interfaces, its (documentation of its) standard compliance, and perhaps illustrating its possible usage by shipping it with code and applications which illustrate the possibilities by using the component (while also teaching how to use it).

Consideration: Component development involves more formal verification process to assure good component quality. Bug fixing in maintenance phase is not trivial as far as components are concerned

Application: More formal approach for component validation and verification combined with component certification should be introduced when applying agile ideas to development of COTS components.

Integration

Table 9: Applying Agile Principles to Integration Process Activity

Agile Principles		Integration
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	P1	Principle is supported by the XP practice Continuous Integration
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	P2	-
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	P3	*Relevant to the whole process.*
Business people and developers must work together daily throughout the project.	P4	Principle is supported by XP practice Continuous Integration
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	P5	*People, not process aspect /Project Management consideration.*
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	P6	-
Working software is the primary measure of progress.	P7	*Relevant to the whole process.*
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	P8	*People, not process aspect /Project Management consideration.*
Continuous attention to technical excellence and good design enhances agility.	P9	-
Simplicity-the art of maximizing the amount of work not done-is essential.	P10	-
The best architectures, requirements, and designs emerge from self-organizing teams.	P11	*People, not process aspect /Project Management consideration.*
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	P12	*Relevant to the whole process.*

Table 10: Integration Process Activity Characteristics for Agile SD, System Development with COTS Components and Component development

Agile Process Activity Characteristics	System Development with COTS Components Process Activity Characteristics	Component Development Process Activity Characteristics
Integration is done continuously so that the system is kept integrated and running most of the time Automated testing supports frequent integration Regular builds are produced which enables quick feedback from the customer	The integration of COTS components into a system is a main part of development activity. General system integration happens during the building process.	Component development involves stricter quality control and a more formal integration process is needed.

System Development with COTS Components

Comment: There are two distinct meanings of "integration" in component-based systems: first, the actual system development means integration of components through the writing of 'glue code' and configuration of components (which is treated in the section Development). Second, there is a build process involved, as in any system development, and this includes a large element of integration. The second meaning is what is treated in this section, although these two types of integration may be difficult to discuss in isolation of each other for component-based systems.

Application: No particular contradictions or constraints exist for applying frequent build process and system integration.

Component development

Comment: No particular contradictions or constraints seem to exist since component development is not significantly different from common application development as far as integration is concerned. With this we mean that whether you develop a "component" or a "system" in-house, you clearly need mature configuration management processes and tools to keep track of changes to your modules etc., but there is nothing particular for the component-based approach.

Consideration: In agile methods integration happens continuously. However, when dependability issues are addressed (such as availability, reliability and performance), the integration process should be controlled, i.e. some explicit integration procedure may be explicitly needed which includes e.g. running performance tests, formal bug reporting etc.

Application: No particular contradictions or constraints exist for applying frequent build process and system integration. However, it can be not so appropriate in cases when some formal integration process is required.

High- level considerations

This section describes common ideas behind some of the agile principles that have relevance to the whole process rather than to particular activities, and which have not been discussed in the previous sections.

Table 11: High-level considerations on applying Agile Principles to COTS-based Development

Agile Principles		System Development with COTS Components	Component development
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	P1	Early delivery can be achieved very efficiently, since prototypes can be built very fast by integrating the existing components. Continuous delivery is feasible, since component-based development enables fast integration and achievement of functions by integration of new components. Greatest value for the customer can be achieved by requirements prioritization by business value.	Early delivery could be problematical; for the component users it is important that the interface does not have to change too often, which is a risk if released (too) early. Continuous delivery is feasible, since in principle the implementation can be changed often while keeping the interface. Continuous delivery could be very smooth for the system developer; depending on the technology used, it could be very simple to replace an old component version with a new without rebuilding (e.g. compiling) the system. There is no straightforward way to achieve the greatest value for the customer as there is usually no single customer(s).
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	P2	Changing requirements in a system made of components will often affect (the selection and/or implementation of) components. To implement a change there are several alternatives in theory: 1) if there is a straightforward mapping from a system requirement to component features, these requirements are easily implemented by some more 'glue code', 2) one or more components need to be replaced by some other(s), or 3) some components need to be improved to accommodate the new requirement. Alternatives 2 and 3 are however often non-trivial, costly and time-consuming in practice.	Late changes to COTS component are limited to 'upgrading' the existing functionality by keeping the old interfaces so that old version of the product doesn't become obsolete.

Agile Principles		System Development with COTS Components	Component development
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	P3	If we refer to the discussion below the principle is connected to short iterations and feedback which both are very easily achievable.	Short iterations and feedback is not so feasible here as it depends on how the customer is emulated. If we have to gather a lot of customer representatives regularly each month or so it can be very difficult task...
Business people and developers must work together daily throughout the project.	P4	This principle is independent of whether components are used or not, and can be applied easily.	This principle is independent of the component-based paradigm, and can be applied easily. This principle is particularly important with business people aiming for reusability. However, for the special case of COTS components, i.e. components offered to a market, there is no clear customer who would specify the requirements. This becomes especially problematic when it comes to the reusability aspect of the component, since it requires that the developer know how the component will be used (customized, adapted etc.). This can be solved e.g. by someone internally who knows the market acting as customer, or involve one or a few important customers during the development process. These customers could represent either some special type of customers (interesting for reusability), or the largest customers, or the average customers.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	P5	This principle is independent of whether components are used or not, and can be applied easily. There is however a risk that the individuals would prefer to develop (implement) functionality rather than glue components together (the "not invented here" syndrome), which need to be addressed with education and organizational culture.	This principle is independent of the component-based paradigm, and can be applied easily. The reusability should be the continuous challenge.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	P6	This principle is independent of whether components are used or not, and can be applied easily. Since the results can be visible very fast, the discussion becomes very concrete and any assumptions can be easily validated.	This can be a problem if components are assumed to be used by as many as possible customers. See P2.

Agile Principles		System Development with COTS Components	Component development
Working software is the primary measure of progress.	P7	(Similar to P3) Working software: This principle is supported by the component-based paradigm itself; by assembling already working components, the first working system can be rapidly built and new functions added by utilizing more features in the existing components, and/or adding new components, and/or extending the 'glue code'.	(Similar to P3) Working software: This principle is important but more complex to apply to a software component. To present working software, the component developer must ensure there is a system that can demonstrate the component capabilities. Such a system could range from being a real, large, complex system developed by a real customer, to a fairly simple system used only for demonstration purposes. Such a system requires additional effort, and introduces additional complexities into the development since synchronization between component development and system development is needed.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	P8	This principle is independent of whether components are used or not.	This principle is independent of the component-based paradigm.
Continuous attention to technical excellence and good design enhances agility.	P9	Independent of the component-based paradigm, can be in line, there are also risks. Component technologies are advanced and are able to guarantee (to some extent) a high quality. There is a risk however that a person without high quality skills can integrate a system and incorporate – and hide – some quality problems into the system.	This principle requires particular skills from developers, to achieve component reusability and usability, and adherence to standards. An important goal for the component developers is that systems built using the component can achieve a good and clean design.

Agile Principles		System Development with COTS Components	Component development
Simplicity – the art of maximizing the amount of work not done – is essential.	P10	Using components (large black boxes of functionality) can result in a clean and simple design; however it is also possible that components have a fairly high lowest level of usage complexity, so that it need to be properly initialized, adapted, its methods called in a certain order etc. even though a very simple functionality is desired. To a certain extent, this may be inherent in the component-based paradigm, but to a large extent this is a challenge for the component developers. This principle is also applicable in component selection process [20]: select the component that has the desired functionality for the moment. However, since component replacement is non-trivial (see discussion in section Requirements) this may be a point where more planning for future requirements is needed, i.e. contrary to the agile principle.	Making components as simple as possible internally is independent of the component-based paradigm. A big challenge is to create a component which is as simple to use as possible, i.e. its interface (in a broad sense) should be as simple as possible. As components and their interfaces evolve, this becomes even more challenging, since there is a difficult tradeoff between strictly extending an interface and breaking or replacing it. By strictly extending an interface, it will be backward compatible, however this typically comes with additional complexity both to build it and to understand and use it (i.e. both internally and externally). By breaking or replacing an interface, all systems using the component need to be modified to be able to use the new component version; the option of supporting both an old and a new interface introduces additional complexities (both externally and internally) and costs (internally) but could be an option for a limited time period.
The best architectures, requirements, and designs emerge from self-organizing teams.	P11	This principle is independent of whether components are used or not.	This principle is independent of the component-based paradigm.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	P12	This principle is independent of whether components are used or not.	This principle is independent of the component-based paradigm.

Principle 1 (P1) has 3 dimensions:

- Early delivery- early feedback, quick wins [12].
- Continuous delivery- continued wins, visibility of where the project goes.
- Delivery of those items that have greatest value for the customer [12].

The first two dimensions are applicable at process level, not process activity level. They mean that software should be developed in short timescales and continuously incrementing the functionality that is released. The first cycle has to be kept as small as possible to allow for early feedback. Several practices support this approach:

- ASD: Adaptive life cycle
- DSDM: Iterative and incremental development is necessary to converge on an accurate business solution.

- FDD: Developing by feature
- XP: Small Releases
- XP: Planning Game- Release planning

Principle 3 (P3) is related to the P1 but focuses on iteration length and feedback. "Early and continuous" in the first principle is more about releases and doesn't specify exactly the length of release cycles. The release cycles should be kept small but because this is not always possible [12]. More frequently on certain points the software should be reviewed and feedback should be provided. For example in XP releases are planned separately from iterations- the iterations are kept as small as possible to produce a working software and to assure early feedback from the user. A single release can be implemented by several iterations. Also in Scrum the iteration length is fixed to 1 month which is not possible to do for a release. The following practices support frequent delivery of working software:

- ASD: Adaptive life cycle
- DSDM: The focus is on frequent delivery of products
- FDD: Regular build schedule
- Scrum: Sprint

Principle 7 (P7) is connected to P3. It can also mean that there is no formal acceptance or status reporting on phase level.

Principle 12 (P12) focuses on the importance of receiving feedback from the team as a way to improve the efficiency of the process. The team should reflect on its "working habits every other week" [12] and continuously adjust itself to be more effective. There are several techniques that are used in agile methods to support the idea:

- Scrum: post-sprint review meeting after each Sprint.
- Crystal Clear: Reflection Workshops.
- ASD (Adaptive System Development): Quality Review: Postmortems

Different Assumptions: Relation to Customers

A fundamental difference in assumptions between agile methods and the usage of COTS is the relation to the customer(s). The agile principles assume that there are one or more customers that initiate the project and for whom the product is created, while COTS products are developed and then offered to an open market with many potential customers. This fundamental difference in the assumptions is the cause of many of the contradictions mentioned earlier, and this need to be addressed. There are two ways, which can be combined, to alleviate the problem outlined earlier:

- Someone internally, who knows the market well, such as marketing people or domain experts, would act as customer in an agile project.
- The component development organization could involve real customers for e.g. requirements gathering and evaluation of various alternatives early during development.

When developing components for a larger market it is the component vendor who finally defines and prioritizes requirements. The goals are not decided ultimately by the customers (or their representatives) but by the one that develops the component. The component developer should of course listen to the customer but it is not the customer who makes the decision. This also means that the contract between component vendor and customer representatives has to be different. Some open

questions that have to be answered are: Can a component developer require an on-site customer? What is the business model for this (i.e. who pays who)? etc.

Test-Driven Development and Component Selection

The general idea of test-drive development (TDD) applied to system development with components would mean that functional tests are specified before implementing a function, after which the 'glue code' for the function is created, followed by test execution. When changes are made, these tests are used for regression testing. The TDD approach can easily be extended to also include component selection: functional tests are specified together with the customer, in parallel with a search for suitable components. There are some established component selection methods where the selection is closely intertwined with requirements specification [3][10][20][23][24][26]. Selection and design also influence each other in both directions: components must be selected which fit the specified architecture, but the availability of components will influence the design; components can for similar reasons with advantage be evaluated and selected in compatible sets simultaneously [9][20][25]. The evaluation of components need to start with some exploration and experimentation to learn the component, but should then mainly consist of the implementation of the features specified by the functional tests. This ensures that the development efforts are kept focused and that the evaluation is relevant. This applies not only to functional testing but also to quality tests. Performance tests would by construction accurately reflect the usage expected in the real system and it is possible to find the relevant limitations and bottlenecks.

The component evaluation can thus be seen as a verification of the suitability of certain components and a certain design as well as the suitability for implementing the system requirements. Verification of the design includes architectural properties and "integrateability", i.e. how well the components integrate in practice. In this way, it is possible to show something to the customer very early in the process. It also becomes possible to involve the customer in the selection decision, if we have, say, three alternative implementations of the same function to show, which is very much in line with agile principles.

Summary

We have provided a systematic, theoretical comparison of agile principles and the fundaments of component-based software development. These two approaches to software development are complementary in several ways, but we have also identified issues which must be carefully considered when applying the agile principles to the development of systems using COTS components, as well as the development of COTS components themselves. More detailed analyses of this data will follow in other publications.

This research should be extended in several ways. First, the same type of analysis can be done for other activities not treated in this report, such as project management, configuration management maintenance and evolution, and documentation. Second, the same kind of analysis can be done for other types of component-based development, such as product line development and architecture-driven development [15]. Third, we believe empirical research is needed to validate, and provide more insight into how serious the issues identified in this study are for organizations in practice, and also identify additional issues and complicating factors. In our own research, we will mainly pursue the third track.

Acknowledgements

We would like to thank Ivica Crnkovic for fruitful discussions during the preparation of this report. This work is partly funded by the Swedish Foundation for Strategic Research.

References

[1] A. Alvaro, E.S. Almeida and S.R.L. Meira, "Software Component Certification: A Survey", In *The 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track*, Porto, Portugal, 2005.

[2] A. Alvaro, R. Land and I. Crnkovic, *Software Component Evaluation: A Theoretical Study on Component Selection and Certification*, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-217/2007-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, 2007.

[3] Carina Alves and Jaelson Castro, "CRE: a systematic method for COTS components Selection", *Proceedings of the XV Brazilian Symposium on Software Engineering (SBES)*, Rio de Janeiro, 2001.

[4] Astels D., *Test-Driven Development: A Practical Guide* , ISBN 0131016490, Prentice Hall, 2003.

[5] Beck K., *EXtreme Programming EXplained: Embrace Change*, ISBN 0201616416, Addison Wesley, 1999.

[6] Beck K., Embracing Change with Extreme Programming, *Computer*, vol.32, no.10, pp. 70-77, Oct., 1999.

[7] Beck K. and Fowler M., *Planning Extreme Programming*, ISBN 0201710919, Addison Wesley, 2000.

[8] Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R. C., Mellor S., Schwaber K., Sutherland J., and Thomas D., *Manifesto for Agile Software Development*, URL: http://agilemanifesto.org/

[9] Jesal Bhuta and Barry Boehm, "A Method for Compatible COTS Component Selection", *Proceedings of the 4th International Conference on COTS-Based Software Systems*, Spain, LNCS, Vol. 3412, Springer, 2005.

[10] L. Chung and K. Cooper, "Defining Goals in a COTS-Aware Requirements Engineering Approach", *Systems Engineering*, Volume 7, Issue 1, pp. 61-83, Wiley, 2004.

[11] Cockburn A., *Crystal Clear: A Human-Powered Methodology for Small Teams*, ISBN 0201699478, Addison-Wesley Professional, 2004

[12] Cockburn A., *Agile Software Development: The Cooperative Game (2nd Edition)*, ISBN 0321482751, Addison-Wesley Professional, 2006.

[13] Santiago Comella-Dorda, John Dean, Edwin Morris, and Patricia Oberndorf, "A Process for COTS Software Product Evaluation", Proceedings of the *1st International Conference on COTS-Based Software System*, Orlando, Florida, Vol. 2255, pp. 86-96, Springer, 2002.

[14] Crnkovic, I., Larsson, M. *Building Reliable Component-Based Systems*, ISBN 1-58053-327-2, Artech House, 2002.

[15] Crnkovic, I., Larsson, S., Chaudron, M., "Component-based Development Process and Component Lifecycle", In *27th International Conference Information Technology Interfaces (ITI)*, IEEE Computer Society, 2006.

[16] DSDM Specification, http://dsdm.com/products/dsdm_version_4_2.asp

[17] J. Fredriksson, R. Land, "Reusable Component Analysis for Component-Based Embedded Real-Time Systems", *29th International Conference on Information Technology Interfaces (ITI)*, Cavtat, Croatia, IEEE, 2007.

[18] Highsmith J., *Agile Software Development Ecosystems*, ISBN 978-0201760439, Addison Wesley Professional, 2002.

[19] Jyrki Kontio, “OTSO: A Systematic Process for Reusable Software Component Selection”, University of Maryland report CS-TR-3478, UMIACS-TR-95-63, December 1995
[20] Land R., Blankers L., *Classifying and Consolidating Software Component Selection Methods*, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-218/2007-1-SE, 2007.
[21] Stig Larsson, *Improving Software Product Integration*, Licentiate Thesis, Mälardalen University Press, 2005.
[22] Lawlis, Patricia K., Mark, Kathryn E., Thomas, Deborah A., Courtheyn, Terry, “A Formal Process for Evaluating COTS Software Products”, *IEEE Computer*, Volume 34, Issue 5, 2001.
[23] Anna Liu and Ian Gorton, “Accelerating COTS Middleware Acquisition: The i-Mate Process”, *IEEE Software*, Volume 20, Issue 2, pp. 72-79, March 2003.
[24] Neil A. Maiden and Cornelius Ncube, “Acquiring COTS Software Selection Requirements”, *IEEE Software*, Volume 15, Issue 2, pp. 46-56, March 1998.
[25] Maurizio Morisio, Carolyn B. Seaman, Victor R. Basili, Amy T. Parra, Steve E. Kraft, and Steven E. Condon, “COTS-based software development: Processes and open issues”, *Journal of Systems and Software*, Volume 61, Issue 3, pp. 189-199, Elsevier, 2002.
[26] Cornelius Ncube and Neil A. Maiden, “PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm”, *Second International Workshop on Component-Based Software Engineering*, Los Angeles, CA, USA, 1999.
[27] Cornelius Ncube, John C. Dean, “The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components”, In *Proceedings of the First International Conference on COTS-Based Software Systems*, LNCS 2255, p176 - 187, Springer-Verlag, 2002.
[28] Perry D., Grishman P., Architecture and Design Intent in Components & COTS Based Systems”, In *Proceedings of the Fifth International Conference on COTS-Based Software Systems (ICCBSS)*, 2006.
[29] Robert Martin C., Micah, Martin, *Agile Principles, Patterns, and Practices in C#*, ISBN 978-0-13-185725-4, Prentice Hall, 2006.
[30] Schwaber K., *Agile Project Management with Scrum*, ISBN 9780735619937, Microsoft Press, 2004.
[31] Sommerville I., *Software Engineering*, 8th edition, Addison Wesley / Pearson Education, 2007.
[32] Stojanovic, Z., Dahanayake, A.N.W., “Component-Oriented Agile Software Development”, In *Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Genova, Italy, 2003.