DESIGN OF A MULTI-AGENT SYSTEM ARCHITECTURE FOR THE SCRUM METHODOLOGY

Vishwaduthsingh Gunga[1], Somveer Kishnah[2] and Sameerchand Pudaruth[3]

[1]TNT Express ICS (Mauritius)
ashvin.gunga@gmail.com
[2]University of Mauritius
s.kishnah@uom.ac.mu
[3]University of Mauritius
s.pudaruth@uom.ac.mu

ABSTRACT

The objective of this paper is to design a multi-agent system architecture for the Scrum methodology. Scrum is an iterative, incremental framework for software development which is flexible, adaptable and highly productive. An agent is a system situated within and a part of an environment that senses the environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future (Franklin and Graesser, 1996). To our knowledge, this is first attempt to include software agents in the Scrum framework. Furthermore, our design covers all the stages of software development. Alternative approaches were only restricted to the analysis and design phases. This Multi-Agent System (MAS) Architecture for Scrum acts as a design blueprint and a baseline architecture that can be realised into a physical implementation by using an appropriate agent development framework. The development of an experimental prototype for the proposed MAS Architecture is in progress. It is expected that this tool will provide support to the development team who will no longer be expected to report, update and manage non-core activities daily.

KEYWORDS

Multi-agents systems, Scrum framework, Agile teams, Software development

1. INTRODUCTION

Software development organisations are facing increasingly pressure to deliver software on time and within budget. Indeed they are looking for agile and efficient methods to develop software due to rapid and often unexpected changes in the business environment (Ionel, 2008). Users' and customers' participation throughout the software development cycle have become the norm rather than a luxury. There is no choice here. Either they adapt or they perish. Stakeholders often have changing needs which results in highly volatile requirements. Furthermore, software applications are getting more and more complex. The client's desire to have early views of the software and maximum added value is minimal time cannot be ignored.

The traditional way of building software is a sequential life cycle. It consists of several phases such as gathering requirements, analysis, design, coding, testing, evaluation and maintenance. The traditional software development methodologies such as the Waterfall Model, the V-Model and the Rational Unified Process (RUP) Model are often associated with common challenges and obstacles leading to unnecessary project delays, overspending or simple total failure. Despite the fact that these methodologies have been used extensively for many years, these models do not always deliver what is desired. Overspending on unnecessary features is very common. It is often every difficult and costly to accommodate new requirements late in the development process. Furthermore, the complex change management procedures requiring rigorous approvals and the fostering on adversarial relationships between people working in the different phases further complicates the scenario.

Agile software development methods have evolved in the mid-1990s and have successfully provided an answer to the weaknesses of these traditional models. With more flexible light-weight processes, agile methods aim at delivering products faster, with higher quality and minimising the risks of project failures by developing higher value features in shorter iterations. Changes in requirements as well as new features are accepted throughout the process and customers and other stakeholders are involved earlier. Working software is used as the primary measure of progress. Agile methods are a family of development processes which include the Scrum framework, XP (eXtreme Programming), DSDM (Dynamic Systems Development Method) and ASD (Adaptive Software Development). Scrum is one of the most popular methods. Scrum framework, sometimes referred to as the Scrum methodology, was developed by Jeff Sutherland and Ken Schwaber in early 1990s and formally announced in 1995 (Keith, 2007).

The main idea behind Scrum is that development is done in cycles of work called sprints. A sprint is a time-box. It has a fixed duration, often between two to four weeks. Scrum emphasises on the various roles (product owner, scrum master and scrum team), ceremonies (release planning meeting, sprint planning meeting, daily scrum meeting, sprint review and retrospective meeting) and artifacts (product backlog, sprint backlog, sprint burndown chart, release burndown chart) to produce potentially shippable products at the end of each sprint. The team reviews the sprint with the stakeholders and demonstrates the software features that have been implemented. A sprint retrospective is conducted where feedback is gathered and incorporated in the next sprint (Sutherland, 2010).

As Scrum is becoming more popular as an increasing number of projects are adopting this framework, there is a growing need for tools and techniques to assist the agile teams. Physical scrum task boards, spreadsheet templates and off-the-shelf suites and packages have so far been the primary tools for assistance in many organisations. However, in certain cases, the weaknesses and limitations of the tools overshadow the benefits. For example, the physical scrum task boards are not suitable for geographically dispersed teams. Spreadsheet templates cannot capture traceability of changes within sprints and ready-made suites are often click-intensive and require high human involvement. As a consequence, it is necessary to explore alternative solutions that provide better support to the scrum practitioners. In the recent past, some research has been conducted to use software agent technologies in SPM (Software Project Management) and at different stages of the software development life cycle.

Software agents are autonomous systems that display a certain degree of autonomy in their environment. They are typically both reactive (responsive to a changing environment) and proactive (goal-directed), and are able to communicate with other agents (often called, multi-agent systems) in order to jointly perform a certain task. Depending on the type and nature of software agents, they are suited to distributed environments, large network systems and mobile devices. The use of software agents in the Scrum methodology is an innovative alternative to current solutions and hence, the aim of this paper is to analyse how software agent technology can contribute in the process for better support and assistance.

There are many tools that have been developed to support software development, namely requirements management repository, project planning, defect tracking, source code repository, releases repository and reporting. Some of these tools are targeted for software development process while others are mainly for agile practitioners. Two essential factors identified by Reel (1999) which are critical for the success of software projects are tracking progress and making smart decisions. Most of the tools provide features to support these two areas but often, project management is more complicated. As Scrum is becoming more popular, there is an unprecedented need for the right tools that can support the teams. The core activities for the teams are to turn the requirements and the product backlog into an executable product. However, Scrum is more than just analysis, design, code and test. It is also about following the process while ensuring that the roles are defined, the activities are followed and the artifacts are produced. These non-core activities can consume significant amount of time during the sprint if appropriate tools are not available.

In practice, the main tools currently used in the industry are scrum physical tasks boards, spreadsheets and templates and web-based software. Each of these may work well in certain scenarios and to some extent, but they pose serious limitations in other cases. For example, the physical scrum boards are restricted to small and collocated teams in a single site while in real-life scenarios, teams tend to be geographically dispersed. The web-based tools are manual driven tools and rely heavily on human intervention. They are considered as tools used for persistence and reporting only but the overheads to keep them updated and in synchronisation are left with the scrum teams. Hence, there is a need to investigate about an alternative mechanism that can better assist the scrum teams. As software agents are being introduced in a number of Software Project Management areas, it is important to understand whether software agents can challenge the existing scrum tools and be one of the innovative alternatives in the future.

This paper proceeds as follows. In Section II, the tools and techniques currently being used in Scrum teams are reviewed. This section also describes how software agents are being used for Software Project Management. The novel architecture for the integration of an agent-based technology in Scrum is described in Section III. Finally, Section IV concludes the paper.

2. LITERATURE REVIEW

This section presents the various tools used to support Scrum teams. The advantages and disadvantages of physical scrum task boards, spreadsheets templates, off-the-shelf tools and application packages are discussed. The use of software agents in Software Project Management is also considered. Finally, this section ends with an analysis of how software agents can be used in Scrum.

2.1 Physical Scrum Board

A physical scrum task board is rated as the most effective format of a sprint backlog by Kniberg (2007). The physical scrum task board is often one large whiteboard within the team's work area. It contains the list of tasks that are planned, in progress or completed within any current sprint. The tasks are represented by post-it notes (also known as sticky notes). These notes are reviewed on a daily basis in the daily stand-up meetings and are moved across the board based on the progress made to date. The board enhances communication and interaction as it is visible to all members of the team as well as encourages spontaneous discussions and meetings. For example, if tasks lower down the board are moving to the right without the ones on the top, it implies that the team is focusing on lower priority items. If there are many unplanned items, the tasks were not identified during the sprint planning meeting and hence, the team is not fully focused. If there are too many tasks in progress but not done, tasks are being done only partially and hence, team is not entirely committed. Despite its numerous advantages, boards have a number of weaknesses. Physical task board is not the ideal solution for offices with limited space and where walls are already occupied for other purposes. Furthermore, a development centre may be working on various projects with different teams. It might be difficult to maintain separate task boards for the different projects (or sprints). Another question that can be raised is how will Scrum work with a physical task board if the team members are not collocated? A physical task board is not the ideal tool if the team members are on different floors of the building, different sites or even in different countries (Collabnet, 2008).

2.2 Spreadsheet Packages and Templates

Spreadsheet packages attempt to provide a better tool for Scrum. The features with these packages offer significant flexibility to maintain the details of the backlog items. Searching, sorting, inbuilt formulas, formatting, graphs and charts and automatic updates increase productivity of the entire team. Pre-defined spreadsheet templates are also a common practice in the scrum world. The spreadsheet stores the details of the product backlog, sprint backlog and related items – fields like backlog item id, short descriptive name, importance and priority and initial estimates (Kniberg, 2007). The product owner owns the product backlog. The product backlog is placed in a shared drive where the entire team can simultaneously access the spreadsheet through the shared enabled feature. The scrum team then agrees how the update of the spreadsheet will occur. These solutions address, to some extent, the limitations of a physical task board. Office space or other physical resources are no more a concern. The team members do not need to be in the same site as long as the spreadsheet is accessible. However, there are still issues like traceability of sprint and related tasks, maintaining thousands of rows in a spreadsheet for large projects, the need to manually update the spreadsheet to keep it up-to-date, the risk of the single document getting corrupted or deleted, the possibility of introducing errors during updates and last but not the least, coordinating the updates in bigger teams. Hence, scrum teams are often reluctant to use spreadsheet as a tool within Scrum.

2.3 Off-the-Shelf Packages

There are hundreds of off-the-shelf tools and application packages with the aim to support agile methodologies and practices. A few of these tools directly support Scrum. Banerjee (2010) have observed more than 125 recommendations for tools that support Agile and Scrum based software development from agile experts all around the world. 53 different tools have been recommended. The top 6 are listed: JIRA and its add-ins like Crucible (Code Review), Bamboo (Continuous Integration) and Confluence (Collaboration), VersionOne, Rally, Mingle, Visual Studio Team System with Scrum Templates and Excel templates from Jeff Sutherland.

In May 2010, the Forester Wave published a paper on “Agile Development Management Tools” to evaluate the software vendors. They found that IBM and MKS led the pack with the best overall current features set (West and Hammond, 2010). The use of electronic tools offers a convenient way for distributed team to work together in Scrum and have access to same information. However, the issues with these electronic tools are not only time-consuming to use – time it takes for pages to load and refresh for each operation – but also, they are often click-intensive whereby a number of mouse clicks are required to get simple operations done. Updates rely heavily on human intervention. The team members often require a significant amount of training and coaching to use the tools effectively and efficiently. Progress of tasks is not clearly visible to the team throughout the day until the next daily stand-up meeting is held. The license cost of any of the commercial software is yet another factor to consider.

2.4 Agents in Software Project Management

Nienaber and Barnard (2007) created a generic model to support all the key areas of Software Project Management (SPM). Based on this model, Sethuraman et al. (2008) developed an agent-based Quality Review Management (QRM) module. Bodea and Niculescu (2007) implemented an agent-based model for levelling project resources. Arauzo et al. (2009) also used a similar mechanism to allocate resources for project tasks. They also proposed a multi-agent system with capabilities to deal with the complexities and constraints inherent in a multi-projects environment. Yen et al. (2001) looked at another direction and attempted to solve the issue of limited ability to support proactive information exchange among team members in existing multi-agent models by developing a multi-agent architecture, called CAST. Licorish et al. (2009) designed and constructed a web-based tool called Agile Social Risk Mitigation Tool (ASRMT), which incorporated a component on personality trait assessment.

2.5 Software Agents and the Scrum Methodology

As we have just seen, much research has been done in integration agents in SPM. However, to our knowledge The researchers have either looked at the project management disciplined in a more general framework or some specific areas of SPM, but more from a traditional management viewpoint. There has been almost no serious attempt to thoroughly and objectively examine Scrum and software agent technology in conjunction with real life cases. The generic framework proposed by Nienaber and Barnard (2007) cannot easily be adapted to work in an agile environment. Scrum uses different roles, artefacts and ceremonies compared to the traditional project management styles. Management in Scrum is done in a more informal manner through the daily stand-up meetings, the sprint planning meetings, the sprint review and the retrospective meetings. However, the auction mechanism proposed by Bodea and Niculescu (2007) and later by Arauzo et al. (2009) can be a potential candidate to manage the activities in Scrum in order to reduce the workload on the Scrum teams. In the Scrum's sprint planning ceremony, the scrum participants agrees on a subset of the product backlog items that will be progressed in the next sprint. The selection is based on a number of criteria such as the business value of the backlog item, the priority, the team's experience and knowledge, the estimated efforts and the dependencies.

3. DESIGN OF A MULIT-AGENT SYSTEM (MAS) ARCHITECTURE FOR SCRUM

A In the previous section, the tools that Scrum teams are currently using were considered. Their limitations were also highlighted. As mentioned earlier, there has been almost no attempt to combine agents in the Scrum methodology. In this section, we proposed a model for Scrum that uses software agent technology to assist Scrum teams.

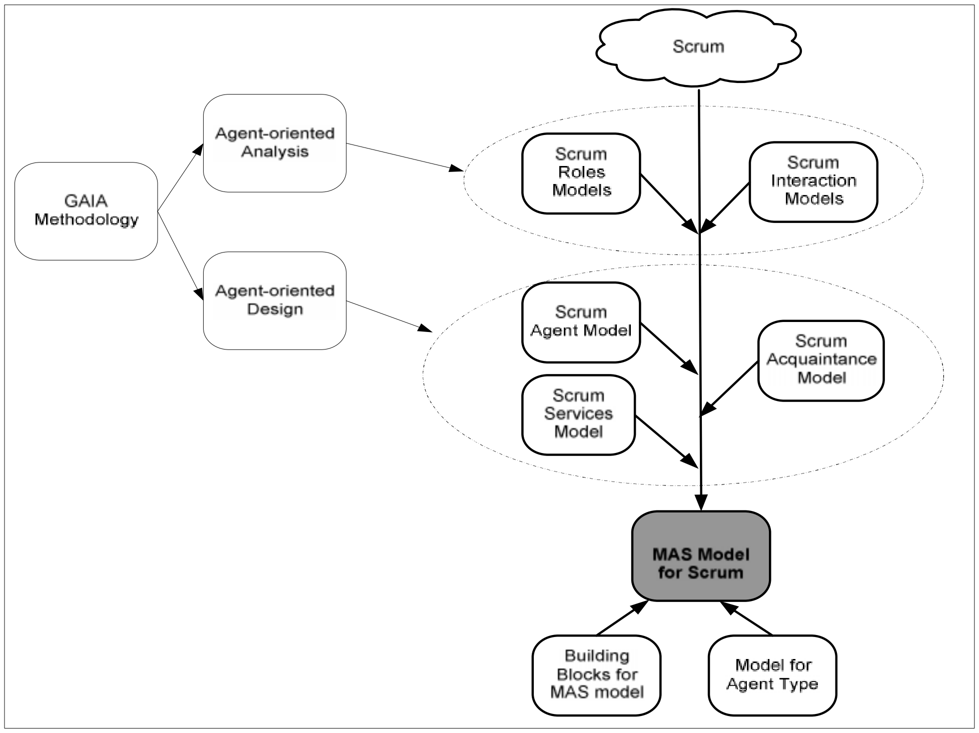


Figure 1. An approach to analysis and design for the MAS Architecture

For the analysis and design of the MAS model for Scrum, the Gaia methodology is used. This allows abstract concepts to be successively translated to more concrete concepts, with increasingly detailed models of the system being designed and constructed. This approach is suitable in the context as no reference to implementation issues is made during analysis.

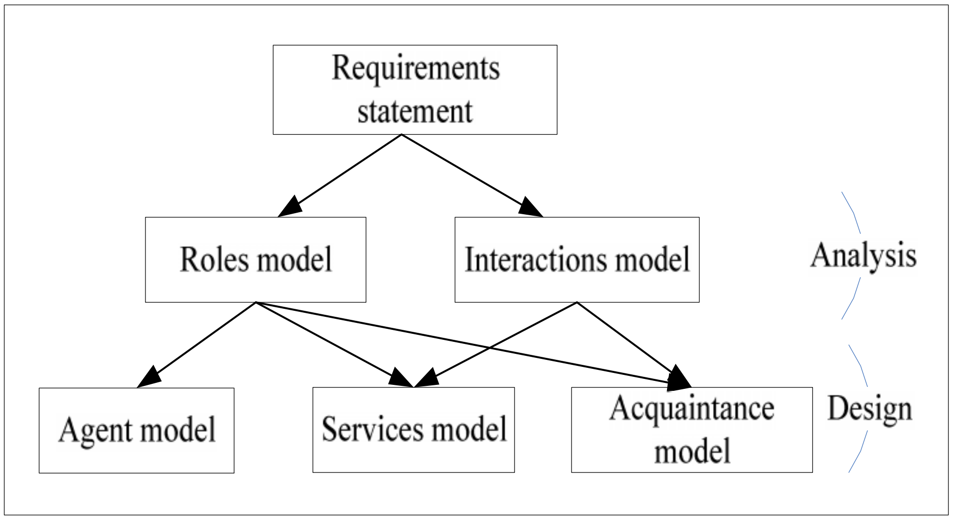


Figure 2. Gaia Models and their Relationships, Wooldridge et al. (2000)

The agent model shows the various agent types involved in the system. The services model presents the functions associated with each agent role and specifies the main properties of the services. Last, the acquaintance model defines the communication links that exist between different agent types.

3.1 Analysis Model for Scrum

In the analysis phase, requirements are converted into roles and interaction models. Models help in understanding the system without going through technical implementation details. The key concepts in an agent-oriented analysis are shown in Figure 3.

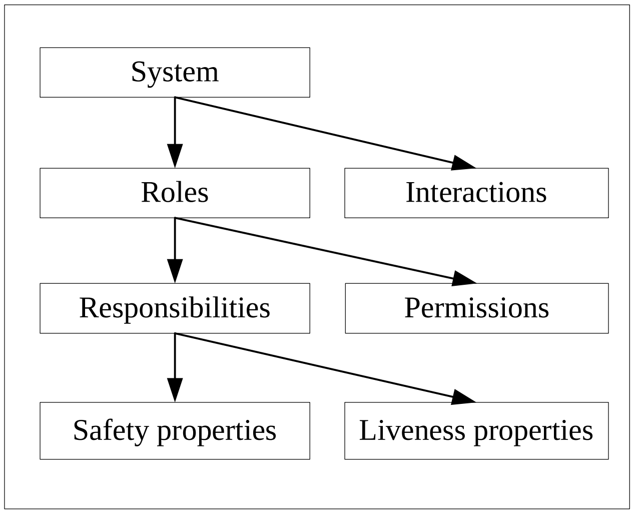


Figure 3. Analysis concepts in Gaia, Wooldridge et al. (2000)

The main abstract entity is the system – the Scrum framework in this case - which relates to the agent-based system. The roles are the key participants within the system. A role is further defined by four main attributes namely responsibilities, permissions, activities and protocols. The responsibilities are the functionalities of a role. A responsibility is further broken down into liveness and safety properties. The liveness properties are the output of agents given certain environmental conditions, while the safety properties are items to be maintained during execution phases. The permissions are the rights associated with a role. The activities are private actions of an agent without interaction with any other agent. Finally, the protocols are the way roles interact with roles in the system.

1) Scrum Role Models

Three roles are involved in the Scrum framework. There roles are taken up by actual individuals or teams in an organisation. There is a natural mapping between the Scrum roles and the roles in the Gaia’s analysis concept. These roles are: the Product Owner, the Scrum Master and the Team. The roles are modelled on the role-schema template described by Wooldridge at al. (2000).

Table 1. Role Model for Scrum Master

Role Schema:	*Scrum Master (SM)*
Description:	*The role involves ensuring that the Scrum process is understood and followed by the entire team. It also helps the team to do its best within its environment.* • *Instantiate a product owner to manage the backlog;* • *Teach product owner how to do his/her job;* • *Ensure that Scrum team adheres to Scrum values, practices and rules;* • *Schedule the sprint planning meeting with the product owner and the team;* • *Communicate sprint details to team and other stakeholders;* • *Setup daily stand-up meeting with team;* • *Ensure that the only questions answered in the daily stand-up meeting are: what I have done yesterday, what are the impediments and what I plan to do today;* • *Log and remove impediments for the team;* • *Ensure that sprint review meeting is held;* • *Notify product owner if sprint goal cannot be achieved;* • *Notify product owner if additional backlog items can be included within the sprint;* • *Notify product owner if backlog items needs to be removed from sprint to meet targets;*
Protocols and Activities:	*InstantiateProductOwner, AdviseProductOwner, AdviseTeamMember, OrganiseSprintPlanningMeeting, CommunicateSprintDetails, ScheduleStandupMeeting, InitiateAndConductStandupMeeting, LogAndRemoveImpediment, OrganiseSprintReviewMeeting, FeedbackSprintRisksToProductOwner, AcceptSprintBacklogItem, RemoveSprintBacklogItem*
Permissions:	• *Read sprint backlog* • *Update impediment list* • *Schedule meeting* • *Generate report*
Responsibilities	
Liveness:	Initiate: (*InstantiateProductOwner. OrganiseSprintPlanningMeeting.CommunicateSprintDetails. ScheduleStandupMeeting.*ProgressStandupMeeting. *OrganiseSprintReviewMeeting)* ProgressStandupMeeting: (*InitiateAndConductStandupMeeting. LogAndRemoveImpediment. FeedbackSprintRisksToProductOwner. AcceptSprintBacklogItem.RemovesSprintBacklogItem)+*
Safety:	*StandupMeeting <= 15 mins* *Sprint Duration <= agreed duration (i.e. 2 weeks)* *Current sprint backlog velocity <= agreed sprint velocity*

Table 2. Role Model for Product Owner

Role Schema:	*Product Owner (PO)*
Description:	*The role involves in managing the product backlog.* • *Create product backlog;* • *Add/Remove user story, feature and new requirement as product backlog item;* • *Initiate Sprint;* • *Provide sprint goal to the team;* • *Review priority for product backlog item;* • *Review business impact for product backlog item;* • *Request for estimates for product backlog items;* • *Explain business needs of product backlog item;* • *Accept/Approve estimates for product backlog item;* • *Approve estimated sprint velocity;* • *Approve changes in sprint backlog;* • *Publish product backlog to everyone and ensures visibility;* • *Cancel sprint;* • *Set date and location for sprint demo;*
Protocols and Activities:	*CreateProductBacklog, AddProductBacklogItem, RemoveProductBacklogItem, UpdateProductBacklogItem, InitiateSprint, CommunicateSprintGoal; UpdatePriority; UpdateBusinessImpact; CommunicateBusinessNeeds; RequestForEstimates, ApproveEstimates; ApproveSprintVelocity; ApproveSprintBacklogItems; PublishProductBacklog; CancelSprint; CommunicateDemoDateAndLocation*
Permissions:	• *Read product backlog* • *Change product backlog / product backlog items* • *Read sprint backlog*
Responsibilities	
Liveness:	Initiate: (*CreateProductBacklog.* ManageProductBacklog *.InitiateSprint.* ParticipateInSprintPlanning). ManageProductBacklog*: (AddProductBacklogItem|* *RemoveProductBacklogItem|UpdateProductBacklogItem|* *UpdatePriority|UpdateBusinessImpact)** ParticipateInSprintPlanning: (*CommunicateSprintGoal.* *RequestForEstimates.ApproveEstimates. ApproveSprintVelocity.* *ApproveSprintBacklogItems.CommunicateDemoDateAndLocation)*
Safety:	*Nil*

TABLE 3. Role Model for Scrum Team

Role Schema:	*Scrum Team (Team)*
Description:	*The role involves in turning subset of the product backlog into a sprint backlog, and subsequently turning the latter into a potentially shippable functionality by the end of the sprint.* • *Attend sprint planning meeting;* • *Break down product backlog items into tasks and sub-tasks;* • *Provide estimates to product backlog items / tasks to product owner for approval;* • *Create sprint backlog;* • *Provide sprint velocity to product owner for approval;* • *Manage sprint backlogs;* • *Update backlog item status;* • *Log work;* • *Attend daily stand-up meeting;* • *Answer the three questions in daily stand-up meeting;* • *Assign product backlog item to team member;* • *Perform sprint demo* • *Participate in sprint review meeting;*
Protocols and Activities:	*AttendSprintPlanningMeeting, BreakDownProductBacklogItem, ProvideEstimates, ProvideSprintVelocity, CreateSprintBacklog, ManageSprintBacklog, UpdateBacklogItem, LogWork, AttendStandupMeeting, AnswerQuestions, AcceptBacklogItem,; PerformDemo, ParticipateInSprintReviews*
Permissions:	• *Read product backlog* • *Read sprint backlog* • *Update sprint backlog*
Responsibilities	
Liveness:	ParticipateInSpringPlanning: (*AttendSprintPlanningMeeting. BreakDownProductBacklogItem.ProvideEstimates. ProvideSprintVelocity.CreateSprintBacklog)* ParticipateInSprint: (*AttendStandupMeeting. AnswerQuestions. AcceptBacklogItem.ManageSprintBacklog.UpdateBacklogItem. LogWork)* ParticipateInReview: (*PerformDemo. ParticipateInSprintReviews)*
Safety:	*Nil*

2) Scrum Interaction Models

In addition to the role models, it is important to capture and represent the interactions between the various scrum roles. This is achieved through the interaction models in Gaia. These models are made up of a set of protocol definitions, one for each type of inter-role interaction. According to Wooldridge et al. (2000), the protocol definition will have the following attributes:

Purpose	: brief textual description of the nature of interaction;
Initiator	: the role(s) that starts the interaction;
Responder	: the role(s) that interacts with the initiator;
Inputs	: information used by initiator as inputs;
Outputs	: information supplied by/to responder during the course of the interaction.
Processing	: brief textual description of any processing during the course of the interaction.

These attributes are then modelled in an interaction diagram as shown in Figure 4.

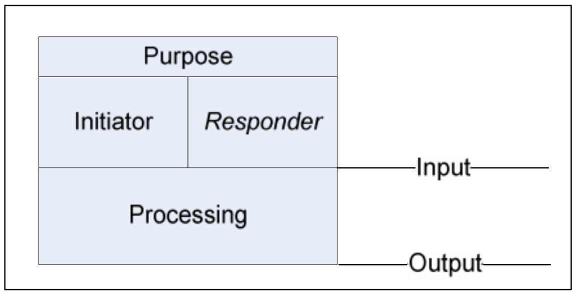


Figure 4. Gaia Interaction Model, Wooldridge et al. (2000)

Considering the descriptions of the three roles in Scrum, there are approximately forty different agent interactions that need to be modelled. Only the interaction models for the sprint planning ceremony are illustrated. The same approach should be adopted to have the models for all the remaining interactions. To develop these models, the sprint planning ceremony needs to be decomposed into further details and repetitive tasks which are candidates for automation are identified.

Sprint Planning

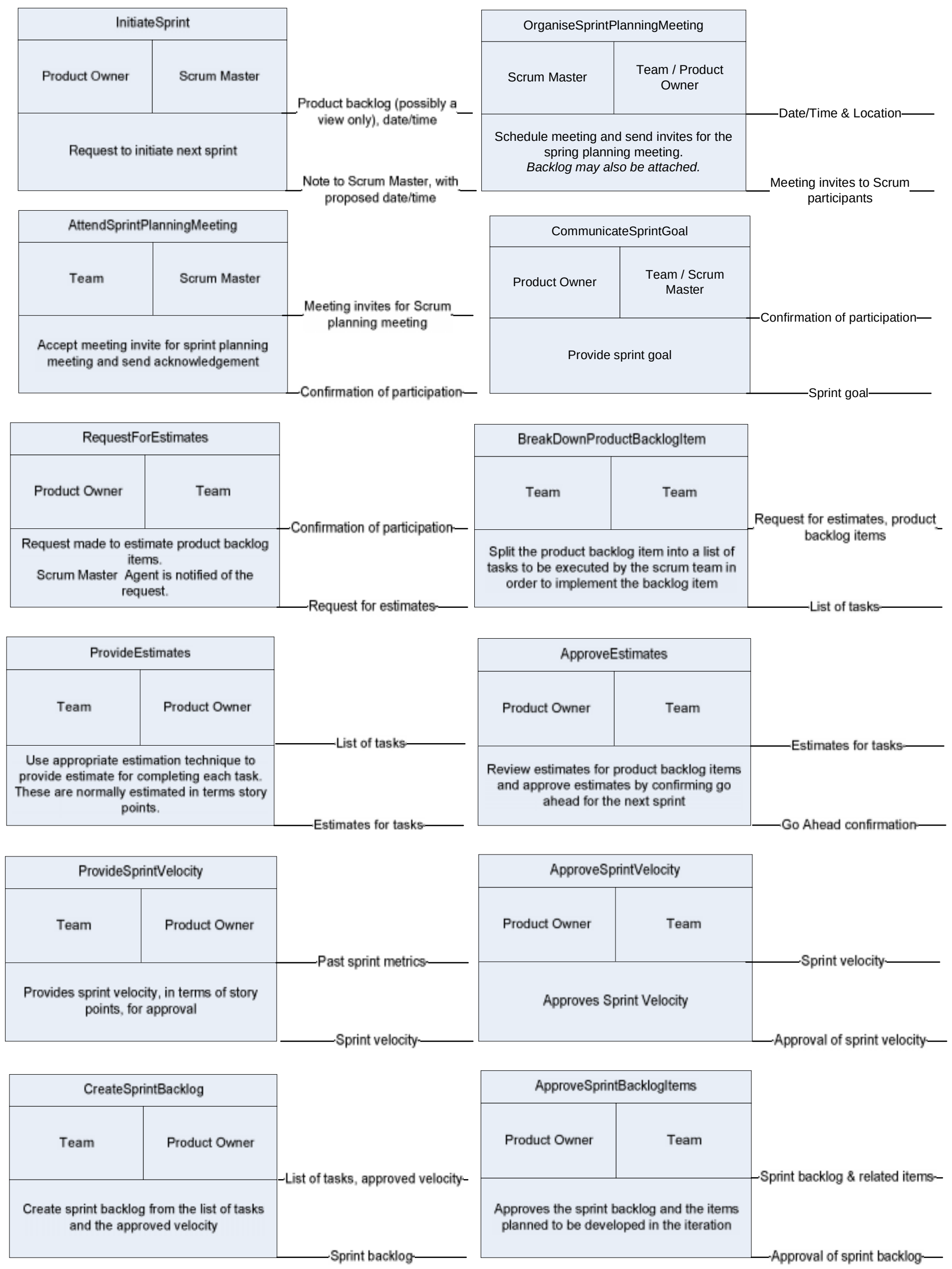
Sprint planning is a very important event in Scrum as the entire sprint success relies on how the sprint planning has been executed. A pre-requisite for the sprint planning meeting is a properly-shaped product backlog. It means that: the product backlog exists, there is one designated product owner, important product backlog items have the correct ratings assigned to them, the product owner understands the backlog items (as there can be new features, new requirements or new user stories), and justifications for having this backlog item in the backlog and its business needs and/or impacts.

The product owner and the entire team (including the scrum master) have to attend the sprint planning meeting. During the meeting, the product owner provides the sprint goal and a brief summary of the product backlog – these are the stories that need to be developed, the scope and importance of the product backlog items. Based on the importance of these stories, the team selects a subset of the backlog items that will be developed for the next sprint. These stories are broken down into tasks and sub-tasks to form the sprint backlog. The sprint backlog items are estimated and use velocity calculations as reality checks. Finally, the team agrees on the completion date as well as the place and time for the daily stand-up meetings. Some of the expected outputs from the sprint planning meeting are: the goal of the sprint, the team members that will be involved in the sprint, breakdown of the product backlog items into tasks/sub-tasks that will form the sprint backlog, the estimates of the tasks/sub-tasks for the product backlog items, the sprint backlog and the planned completion date of the sprint.

The scrum master then communicates the sprint details to all the other stakeholders.

Sprint Planning Interaction Models

The interaction models are presented in Figure 5, as shown below.


InitiateSprint
Product Owner
Scrum Master
Request to initiate next sprint
Product backlog (possibly a view only), date/time
Note to Scrum Master, with proposed date/time
OrganiseSprintPlanningMeeting
Scrum Master
Team / Product Owner
Schedule meeting and send invites for the spring planning meeting.
Backlog may also be attached.
Date/Time & Location
Meeting invites to Scrum participants
AttendSprintPlanningMeeting
Team
Scrum Master
Accept meeting invite for sprint planning meeting and send acknowledgement
Meeting invites for Scrum planning meeting
Confirmation of participation
CommunicateSprintGoal
Product Owner
Team / Scrum Master
Provide sprint goal
Confirmation of participation
Sprint goal
RequestForEstimates
Product Owner
Team
Request made to estimate product backlog items.
Scrum Master Agent is notified of the request.
Confirmation of participation
Request for estimates
BreakDownProductBacklogItem
Team
Team
Split the product backlog item into a list of tasks to be executed by the scrum team in order to implement the backlog item
Request for estimates, product backlog items
List of tasks
ProvideEstimates
Team
Product Owner
Use appropriate estimation technique to provide estimate for completing each task. These are normally estimated in terms story points.
List of tasks
Estimates for tasks
ApproveEstimates
Product Owner
Team
Review estimates for product backlog items and approve estimates by confirming go ahead for the next sprint
Estimates for tasks
Go Ahead confirmation
ProvideSprintVelocity
Team
Product Owner
Provides sprint velocity, in terms of story points, for approval
Past sprint metrics
Sprint velocity
ApproveSprintVelocity
Product Owner
Team
Approves Sprint Velocity
Sprint velocity
Approval of sprint velocity
CreateSprintBacklog
Team
Product Owner
Create sprint backlog from the list of tasks and the approved velocity
List of tasks, approved velocity
Sprint backlog
ApproveSprintBacklogItems
Product Owner
Team
Approves the sprint backlog and the items planned to be developed in the iteration
Sprint backlog & related items
Approval of sprint backlog

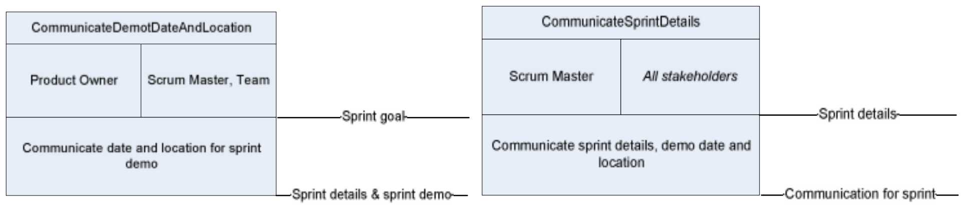

CommunicateDemotDateAndLocation
Product Owner
Scrum Master, Team
Communicate date and location for sprint demo
Sprint goal
Sprint details & sprint demo
CommunicateSprintDetails
Scrum Master
All stakeholders
Communicate sprint details, demo date and location
Sprint details
Communication for sprint

Figure 5. Interaction Models for Sprint Planning

3.2 Design Models for Scrum

The objective of the classical design process is to turn the analysis models into more concrete models with adequate low level of abstraction to allow easy implementation. The Gaia methodology takes the design stage one step further as it is also concerned with how the society of agents will operate to realise the goals of the system. Moreover, it reviews what each agent will require to achieve its own objectives (Wooldridge et al. 2000). In the agent-oriented design process, three design models are generated from the analysis models as depicted in Figure 2, namely the agent model, services model and the acquaintance model.

1) Scrum Agent Model

The first step in the design process is to document the various types of agents that will be used in the Scrum framework. These types are then instantiated as agent instances during execution time. Often, there is a direct one-to-one relation between the roles and the types. The different roles identified in the analysis process are the Scrum Master, the Product Owner and the Team. The agent model for Scrum will be defined using a simple agent type tree. The leaf nodes correspond to the Scrum roles. The other nodes correspond to the agent types. The agent types are annotated using the instance qualifiers in Table 4. The agent model for the Scrum framework is modelled in Figure 6. For the team agent, an assumption of a minimum of three and a maximum of ten team members are involved.

Table 4. Instance Qualifiers, Wooldridge et al. (2000)

Qualifier	Meaning
n	there will be exactly n instances
m..n	there will be between m and n instances
*	there will be 0 or more instances
+	there will be 1 or more instances

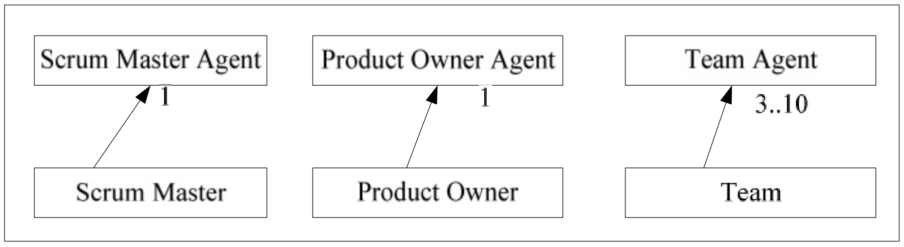


Figure 6. Scrum Agent Model

2) Scrum Services Model

The next step in the design process is to identify the services that are associated with each agent role. In the object oriented world, a service is as simple as a method. In the agent-oriented world, the service is a single block of activity in which an agent will commit. These services are not necessarily available to other agents. For each service, the inputs, outputs, pre-conditions and post-conditions needs to be identified. The service model is directly derived from the roles and interaction models. The inputs and outputs are from the role protocols while the pre- and post-conditions are from the safety properties of a role. The planning of the sprint ceremony is modelled in Table 5.

Table 5. Scrum Services Model – Plan Sprint Ceremony

Services	**Input**	**Output**	**Pre-conditions**	**Post-conditions**
Initiate sprint	Product backlog	Request message	Product backlog available	Request message delivered
Organise sprint planning	Sprint date and time	Message to team members	Team members are available	Meeting scheduled
Attend sprint planning meeting	Sprint data and time and message for sprint planning	Confirm attendance	Team member is free and available	Acknowledgement sent
Communicate sprint goal	Sprint goal	Message to team members	Team members accepted invite	None

3) Scrum Acquaintance Model

The Acquaintance Model in Gaia's design phase shows the different communication links that exist between the different agent types in Scrum. With the model, it is easier to identify if there are any potential communication bottlenecks between the agent types. The model is represented as a directed graph. The nodes on the graph represent the different agent types. The arc, say from a to b, indicates that a will send messages to b, but not necessarily vice-versa. The acquaintance model for Scrum is shown in Figure 7. As there are only three agent types directly derived from the Scrum roles, communication can be between any of the three agents in a two-way direction.

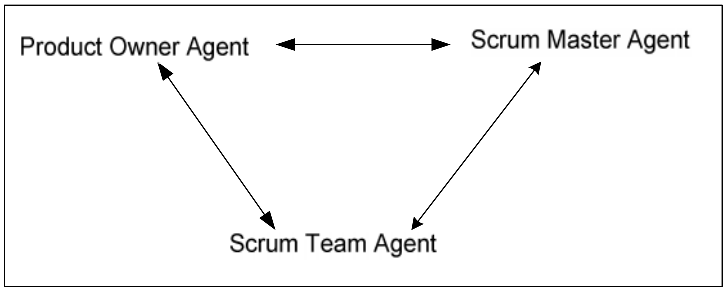


Figure 7. Scrum Acquaintance Model

3.3 Proposed Multi-Agent System (MAS) Architecture for Scrum

Figure 8 presents the MAS Architecture to support Scrum. This models acts as the design blueprint and baseline architecture for solutions aiming at assisting the Scrum team using agent technology. This model is tailored to support the entire Scrum methodology contrary to existing tools and techniques which are often limited to only the analysis and design phases. Furthermore, this model provides a conceptual view of the system. It is therefore independent of implementation issues. The use of a multi-agent system has been chosen for various reasons. The agents in the system can be launched from different workstations over the network and hence provide flexibility to geographically dispersed teams. More agents for the scrum team can be added easily into the model without changing the architecture. Each agent is managed independently in an autonomous way but agents are able to cooperate with each other to achieve the common sprint goals.

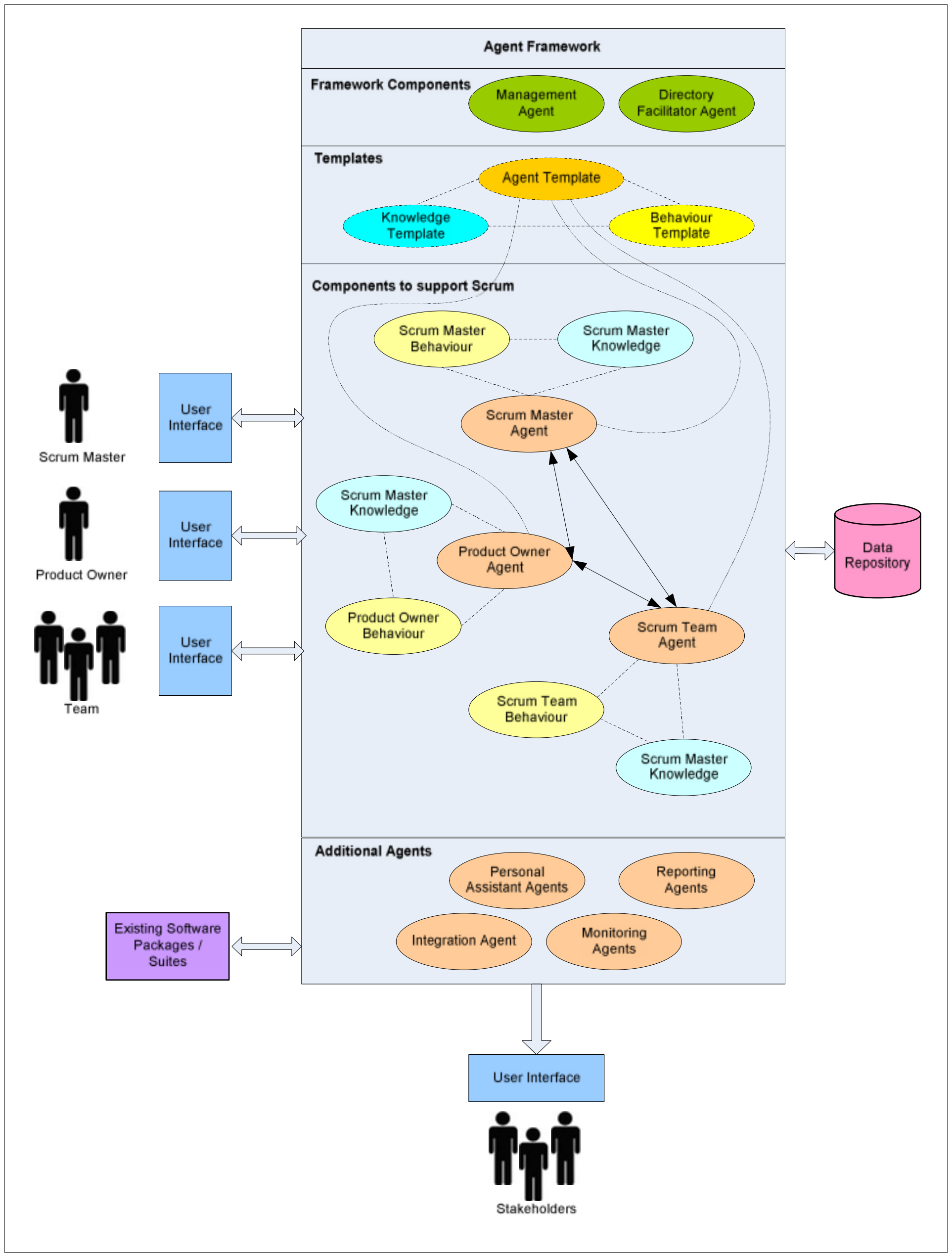


Figure 8. Proposed MAS Architecture for Scrum

Table 5: Components in the MAS Architecture for Scrum

Component	**Description**
Agent Framework	Provides infrastructure to develop, deploy, run and maintain agent-based solutions (e.g. AgentService, JADE).
Framework Components	Components that are often provided part of the agent development framework.
Management Agent	Manages other agents in the system by tracking the instantiation and disposal of agent instances during execution time.
Directory Facilitator Agent	Provides yellow pages service to the agents in the system and provide details of the difference services being offered by other agents. Any agent requiring a particular service can check against the yellow page facility to identify the agents providing such services.
Templates	Provide the default pattern and baseline implementation to the types, behaviors and knowledge components.
Agent Template, Behaviour Template, Knowledge Template	Represent templates implementation for the agent type, agent behavior and agent knowledge.
Scrum Master Agent, Product Owner Agent and Scrum Team Agent	Represent agent type for Scrum Master, Product Owner and Scrum Team. Each agent type follows the Agent Template.
Scrum Master Behavior, Product Owner Behavior and Scrum Team Behavior	Reflect the protocols for each scrum role and provides the list of services that each agent provides in the system.
Scrum Master Knowledge, Product Owner Knowledge and Scrum Team Knowledge	Hold the necessary rules to be able to implement the protocols and related services for each scrum role.
User Interface	Allows the scrum participants and stakeholders interact with the system through input and output.
Data Repository	Represents the main data store for all information for Scrum (e.g. product backlog items, sprint backlog items, tasks allocations, task progress etc). It can be anything ranging from simple text files or XML files to relational databases.
Additional Agents	Agents that are not directly related to the Scrum process but will be required to support the solution.
Reporting Agent	Uses the information from the repository and provides reporting facility by present the information into appropriate formats (e.g. product burndown charts) to the stakeholders and scrum participants.
Personal Assistant Agents	Assist individual team members in specific tasks e.g. sorting of assigned tasks based on priority.
Monitoring Agent	Monitors and controls the tasks and activities of the agents in the system.
Integration Agent	Provides integration facility to existing solutions by exchanging information (e.g. in XML) to external systems

Each agent type in the MAS is associated with a behaviour object and a knowledge object. The agent type extends the agent template and acts according to the template. The agent type is instantiated during runtime to create the agent instances. The knowledge base of the agent type is represented by the knowledge object. For example, the knowledge object for the product owner holds the details of the product backlog, while one for the scrum team contains details of the sprint backlog items. The agent template ensures that its instances have exclusive access to the knowledge objects and deals with any concurrency issues.

The agent framework handles the persistence of these knowledge objects in a transparent way to the application developers. The behaviour object implements the abilities and services of the agent and holds all agent computational logic. The behaviour objects in the MAS Architecture are implemented by extending the Behaviour Template. The association of the knowledge and behaviour object makes the agent types autonomous in the system to achieve the common sprint goals. The agents use the directory facility to identify other agents in the system and initiate conversations to pass messages and communicate with each other. The model for the agent type presented in the proposed MAS Architecture is shown in Figure 8.

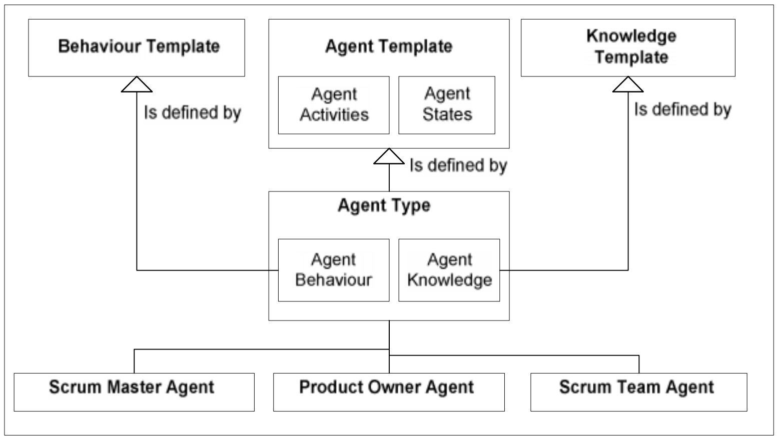


Figure 8. Model for Agent Type in the proposed MAS Architecture for Scrum

4. CONCLUSIONS

The aim of this paper was to present a novel multi-agent system architecture that can be used for Scrum. The autonomous nature of agents provides the possibility for scrum participants to delegate the non-core activities to the system. As a consequence, the scrum team can focus on software development during a sprint rather than on managing tasks. The reporting agent in the MAS architecture provides the ability to report information on more regular basis without being dependent on the team. The proposed MAS architecture is also suitable in scenarios where the teams are geographically dispersed. The scrum team members may be located in different buildings on sites across the country or across continents. The work in this paper is a first attempt to combine software agent technology in the Scrum process. Thus, it provides a completely new direction to research in such disciplines. The development of an experimental prototype to validate our proposed model is in progress.

REFERENCES

[1] ARAUZO, J.A., GALAN, J.M., PAJARES, J. and LOPEZ-PAREDES, A., 2009. Multi-agent technology for scheduling and control projects in multi-project environments. An Auction based approach. Inteligencia Artificial 42 (12-20).

[2] BANERJEE, U., February 17, 2010. Agile Tool – Expert Recommendation. Accessed 20 October 2012, https://udayanbanerjee.sys-con.com/node/1289479.

[3] BODEA, C. N. and NICULESCU, C. S., 2007. Improving Resource Leveling in Agile Software Development Projects through Agent-Based Approach. Journal of Applied Quantitative Methods 2 (2).

[4] COLLABNET, 2008. Scrum with a Physical Taskboard. Accessed 9 May 2013, http://scrummethodology.com/scrum-with-a-physical-taskboard.

[5] FRANKLIN, S. and GRAESSER, A., 1996. Is it an Agent or just a Program?: A Taxonomy for Autonomous Agents. ECAI '96 Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages. Pages 21-35.

[6] IONEL, N., 2008. Critical Analysis of the Scrum Project Management Methodology. Proceedings of the 4th International Economic Conference on European Integration – New Challenges for the Romanian Economy. May 30-31, Oradea, 435-441.

[7] KEITH, K., Feb 2007. Scrum Rising: Agile Development could save your studio. Game Developer, 14(2), pp. 22-26.

[8] KNIBERG, H., 2007. Scrum and XP from the Trenches – How do we Scrum. C4 Media Inc., InfoQ Enterprise Software Development Series, ISBN 978-1-4303-2264-1.

[9] LICORISH, S., PHILPOTT, A. and MACDONELL, S. G., 2009. Supporting Agile Team Composition: A Prototype Tool for Identifying Personality (In)compatibilities. ICSE Workshop on Cooperative and Human aspects on Software Engineering.

[10] NIENABER, R. and BARNARD, A., 2007. A Generic Agent Framework to Support the Various Software Project Management Processes. Interdisciplinary Journal of Information, Knowledge and Management, Vol. 2.

[11] REEL, J. S., 1999. Critical Success Factors in Software Projects. IEEE Software, 16(3), pp. 18-23.

[12] SETHURAMAN, A., YALLA, K.K, SARIN, A. and GORTHI, R.P., 2008. Agents Assisted Software Project Management. COMPUTE '08 Proceedings of the 1st Bangalore Annual Compute Conference, Article 5.

[13] SUTHERLAND, J., July 2010. Scrum Handbook. The Scrum Training Institute. Accessed 15 July 2011, http://jeffsutherland.com/scrumhandbook.pdf.

[14] WEST, D. and HAMMOND, J. S., 2010. The Forrester Wave: Agile Development Management tools, Q2 2010. Accessed on 12 August 2011, http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=RAL14023USEN.

[15] WOOLDRIDGE, M., JENNINGS, N. R. and KINNY, D., 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, Vol. 3, Issue 3.

[16] YEN, J., YIN, J., IOERGER, T. R., MILLER M. S., XU, D. and VOLZ, R. A., 2001. CAST: Collaborative Agents for Simulating Teamwork. Proceedings of the 17th International Joint Conference on Artificial Intelligence, Vol. 2, pp. 1135-1142.