A Context-Driven Software Comprehension Process Model

Wen Jun Meng[1], Juergen Rilling[1], Yonggang Zhang[1], René Witte[1], Sudhir Mudur[1], Philippe Charland[2]
Department of Computer Science and Software Engineering [1]
Concordia University, Montreal, Canada
{w_meng,rilling,yongg_zh,rwitte,mudur}@cse.concordia.ca
System of Systems Section[2]
Defence Research and Development Canada
Val-Bélair, Québec, Canada
philippe.charland@drdc-rddc.gc.ca

Abstract

Comprehension is an essential part of software evolution. Only software that is well understood can evolve in a controlled manner. In this paper, we present a formal process model to support the comprehension of software systems by using Ontology and Description Logic. This formal representation supports the use of reasoning services across different knowledge resources and therefore, enables us to provide users with guidance during the comprehension process that is context sensitive to their particular comprehension task. As part of the process model, we also adopt a new interactive story metaphor, to represent the interactions between users and the comprehension process.

Keywords: *Software evolution, program comprehension, process modeling, story metaphor, ontological reasoning*

1. Introduction

Program comprehension is a major factor in providing effective software maintenance and enabling successful evolution of computer systems [8]. Some estimate that up to 50% of the maintenance effort is spent understanding source code [8]. This significant comprehension effort is due to several factors, e.g. differences among comprehension tasks and their specific settings [8, 13]. These variations lead to a non-well defined multi-dimensional problem space that creates an ongoing challenge for both the research community and tool developers. The situation is further complicated by the non-existence of comprehension process models that can be applied to guide maintainers while performing comprehension tasks. Current program comprehension research focuses mainly on developing better techniques and tools to tackle specific aspects of the comprehension problem. Moreover, these techniques and tools are commonly not integrated with each other, due to a lack of integration standards or difficulties to share services among tools.

There has been little work in examining how different comprehension tools work together for end users [10, 16, 25] and how they can collaboratively support a specific program comprehension task. Maintainers are typically left with no guidance on how to complete a particular comprehension task. Our research aims to overcome this lack of context sensitivity by introducing a formal process model that stresses an active approach to guide users (software maintainers and developers) while solving a comprehension task. Within our process model, the basic elements and their major interrelations are formally modeled by Ontology and Description Logic (DL) [28], and the process behavior is modeled by a novel interactive story metaphor. Our approach differs from existing work on comprehension models [1,2,4], tool integration [20, 25], and task specific process models [11,25] in several aspects:

1. Formalization of a context-sensitive comprehension process that is based on Ontology and Description Logic representation. The goal is to integrate different comprehension aspects, like user expertise, reverse engineering, as well as comprehension tools and techniques within a common process model.
2. The ability to reason about information from the ontological representation to be able to provide an *active and context sensitive* support in guiding the comprehension process itself.
3. Introduction of a novel interactive story metaphor, typically found in the multimedia and film making domain, to model the dynamic interaction aspects between users and the comprehension process.
4. Providing an open environment that allows automation in guiding maintainers during the comprehension process.

Second International IEEE Workshop on Software Evolvability (SE'06)
0-7695-2698-5/06 $20.00

The remainder of the paper introduces in Section 2 the motivation for our research. Section 3 presents the relevant research background. Section 4 describes in detail the context driven program comprehension process. In Section 5 implementation and validation issues are described. In Section 6, related work is discussed, followed by conclusions and future work in Section 7.

2. Motivation

One of the major challenges of modeling program comprehension processes is its multi-dimensional problem domain, requiring interactions among different information and knowledge resources. Informally, the goal of our comprehension process model is similar to models used in other application domains, like Internet search engines (e.g. Google[1]) or online shopping sites (e.g. Amazon[2]). Common to all these applications is that they utilize different information resources to provide an active, customized guidance identifying resources and information relevant to a client's specific needs. Our research is not only motivated by this need to synthesize these different information and knowledge resources utilized within a formal framework, but also to provide maintainers with a context during the program comprehension process itself. The formalization of the process model enables the use of additional automated reasoning services to guide programmers in acquiring relevant knowledge across different information resources and hierarchies.

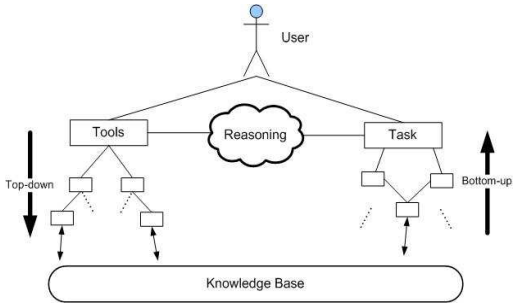

Figure 1 Conceptual Model

For maintainers performing a particular comprehension task, it is not uncommon to utilize and interact with various tools (e.g. parsers, debuggers, source code analyzers, visualization tools, etc.), due to the required pre/post processing to achieve a particular task. Identifying these often transitive relationships among information resources is a challenge. Within our approach we support automated reasoning across these different information resources (e.g. domain knowledge, documents, user expertise, software, etc). Figure 1 shows a simplified illustration of this process by restricting the available information resources to tools and tasks:

(1) Which tools might directly/indirectly be required to perform a particular comprehension task top-down?
(2) Given a current knowledge level (bottom-up), what are the potential (direct/indirectly) related tasks that can be performed?

For the process model to be accepted by end-users, it is essential to provide a supporting framework that captures the context in which the process model is applied. For the contextual representation, we adopt an intuitive visual metaphor – a story driven approach. The story representation addresses three major issues:

(1) A metaphor that is familiar to users.
(2) A context that matches closely a comprehension process and therefore, can be used in actively guiding users while solving comprehension problems.
(3) Stories can be expressed through different media, e.g. text, images, animation or other multi-media techniques.

Our approach differs from existing work by providing a uniform ontological representation of the different information resources, including the context-sensitive user interaction with the comprehension process and the ability to reason across these knowledge resources.

3. Background

This chapter introduces literature relevant to both process modeling and the technical foundations for our comprehension approach.

3.1. Program Comprehension Process

A mental model describes a maintainer's current understanding of a software system. Mental models are formed through the use of cognitive models that describe both the cognitive processes and information structures needed to create a mental model [13]. Program comprehension is typically referred to as the process involved in constructing an appropriate mental model of a software system to be maintained [1, 2]. In the past decades, several cognitive models have been developed to explain how maintainers comprehend source code. Bottom-up [1], top-down [2], and integrated [4,8] are the three major theories of program comprehension that try to model both the activities and the process involved in creating the mental models for comprehension tasks.

[1] www.google.com
[2] www.amazon.com

There exist various aspects affecting program comprehension [8, 13, 16], making it an inherently complex and difficult problem to address. Some of the major issues affecting the comprehension process are the user's comprehension ability (e.g. experience, knowledge background), the characteristics of the software system to be comprehended (e.g. its application domain, size and complexity), the comprehension task to be performed (e.g. maintenance, reverse engineering, architecture recovery), as well as the tools and software artifacts (e.g. source code, documentation) available to support the comprehension process. The goal of a process model has to be both, the integration of resources affecting the comprehension process and the ability to guide users during the comprehension process itself. Existing research in modeling program comprehension has focused on describing the comprehension process in the context of a specific task domain, e.g. maintenance, reverse engineering.

Historically, software lifecycle models and processes have focused on the software development cycle. However, with much of a system's operational lifetime cost occurring during the maintenance phase, this should be reflected in both the development practices and process models supporting maintenance activities. One approach is to focus on the *software maintenance* process (including program comprehension) being a major part of a total system life cycle/process models, as suggested for example in [19, 29]. These process models focus mainly on the creation of additional software artifacts during software development to support future maintenance efforts.

Reverse engineering, a sub process of software maintenance, focuses on the extraction of information from the source code or documentation in order to reconstruct higher level abstractions and knowledge from a subject system [29]. Reverse engineering in a typical setting is performed in an ad hoc manner, using simple query tools, combined with a significant effort involved by the user to interpret the retrieved or abstracted information [10].

Other semi-automatic processes can be found for example in *architectural recovery* [11], which can be seen as a specific reverse engineering task. In [11], the recovery process is composed of six general phases: definition of architectural concepts, extraction of a source code model, abstraction, improvement of architecture documents, analysis of the extracted architecture and reorganization of source code.

It is generally accepted that even for more special instances of a comprehension task like architectural reconstruction, a fully-automated process is not feasible [11]. Common to all these existing models is:

- They provide only general descriptions of the steps involved in a process, without describing any details or guidelines on the context of these steps.
- The use of existing information resources (e.g. user expertise, source code information, tools) to construct a mental model of a program. However, they lack the necessary formalism to integrate these resources in a uniform model and the ability to infer additional knowledge.
- They lack an active, context-driven guidance during the comprehension process using an intuitive visual metaphor to help reducing a user's cognitive load.

Within our process model, we support existing cognitive models [1,2,4,8] in creating a mental model of systems to be comprehended. In our approach, we provide a formal representation that integrates these information resources and allows reasoning across these resources. Furthermore, we extend these models with an additional context sensitive support, providing the maintainer with guidance on the use of different information resources to accomplish a particular task.

3.2. Ontology and Description Logic

Intuitively, "software comprehension" refers to activities that humans perform: understanding, conceptualizing, and reasoning about software. In this regard, a crucial aim of providing support for software comprehension is to assist and improve human thinking processes. Research in cognitive science suggests that mental models may take many forms, but the content normally constitutes an ontology [27]. Ontologies are often used as a formal explicit way of specifying the concepts and relationships in a domain of understanding [26].

Description Logic (DL), a knowledge representation formalism, is used as a standard ontology language. DL is a major foundation of the recently introduced Web Ontology Language (OWL) recommended by the W3C [24]. DL represents domain knowledge by first defining relevant concepts (sometimes called a class or TBox) of the domain and then using these concepts to specify properties of individuals (also called an instance or ABox) occurring in the domain. Basic elements of DL are atomic concepts and atomic roles, which correspond to unary predicates and binary predicates in First Order Logic. Complex concepts are then defined by combining basic elements with several concept constructors. For example, given atomic concepts: *Method* and *PublicModifier* as well as the atomic role: *hasModifier* (refers to a method that has a modifier), the concept of *PublicMethod* can be defined as follows:

$$\text{PublicMethod} \equiv \text{Method} \sqcap \exists \text{hasModifier.PublicModifier}$$

DL allows for precise characterization of concept taxonomy in a domain, by defining subsumption relationships between concepts. For instance, $\mathsf{Field} \sqsubseteq \mathsf{Variable}$ specifies that *Field* is the sub-concept of *Variable*, which means all instances of the concept *Field* are also instances of the *Variable* concept.

Individuals and their relationships in the domain are specified as instances of the concepts and corresponding roles as well. Take the instance *toString* for example, it has *public modifier*. The DL expressions for *toString* are as follows:

```
toString:Method,(toString,public):hasModifier
```

Based on these given facts, a DL reasoner can automatically infer that method *toString* is a public method. This reasoning is rather simple, more complicated reasoning services can be supported by a suitable DL reasoning system. The Racer [22] system is a state-of-the-art ontology reasoner that has been highly optimized to support very expressive DLs. Typical services provided by Racer include terminology inferences (e.g. concept consistency, subsumption, classification and ontology consistency checking) and instance reasoning (e.g. instance checking, instance retrieval, tuple retrieval, and instance realization). For a more detailed coverage of DLs and Racer, we refer the reader to [22, 28].

3.3 Story

Stories are an instinct, implanted universally in the human minds [17]. They are widely used to convey information, cultural values, and experience [15]. It is suggested that humans build cognitive structures that represent the real events in the their lives using models similar to the ones used for narrative story (Table 1), in order to better understand the world around them [6]. People usually find it is easier to understand information in the context of a story instead of serial lists [15].

In software engineering, the notion of story has already been used in different areas. In Extreme Programming, stories are used as a communication media to elicit requirements. In UML, a scenario diagram is a descriptive story of a use case, detailing what happens between users and the system for a specific function. The sequence diagram can also be viewed as a more formal story representation of system behavior, delineating how operations are carried out -what messages are sent and when, and it emphasizes the time ordering of message passing among objects.

A story itself is a narrative, which in its simplest form provides a temporally ordered sequence of events [7]. Storytelling can be applied by scripting a story of the complete system at design-time or by generating stories dynamically on a per-session basis [23]. A system that can generate stories dynamically is capable of adapting the story narrative to the user's preferences and abilities. Stories themselves can be represented in various ways like using text, image, animation, etc.

Table 1 Story Model [5]

Story Model		Description
Theme		Overall message, concept, or essence of a story; ties every structure and dynamic element.
Genre		Establishes an author's overall attitude, which casts a background on all other considerations.
Characters	Protagonist	Main story character - driver of the story: the one who forces the action to resolve the problem and reach the ultimate goal.
	Impact Character	Might be an antagonist (desire to let the problem grow), guardian (functions as a helper/teacher, a protective character who eliminates obstacles and illuminates the path ahead), or contagonist (works to place obstacles in the path of the Protagonist, and to lure it away from success) etc.
Settings		Place, time, weather conditions, social conditions, or mood or atmosphere etc.
Plot		A series of logic events that develop a story; details the order in which the elements must occur within that story.
Throughline		The point of view and growth of the main character within the story world.
Interaction		What occurs between characters (or even ideas) is presented. Interaction can occur as connections and disconnections.
Conflict		Everything that prevents or gets in the way of the protagonist in achieving the goal.
Climax /Resolution		At this point the obstacles are no longer a threat and any conflict disappears.

4. Modeling Program Comprehension

Our program comprehension model presented in this research is based on three major criteria:

- *Simplicity:* The process model itself must be simple and easy to apply. It should not introduce any unnecessary new complexity to the end user.
- *Active guidance:* A limitation of existing software process models (e.g. RUP [14]) is that these models are passive due to the informal description of the processes, by focusing only on the description of the different process artifacts and notations. This informal representation does not allow for an active guidance on what artifact or notation can/should be applied within the current process.
- *Flexibility and extendibility:* The process model has to be able to adapt to every changing comprehension tasks and their settings, and it also needs to support its own evolution.

A model is essentially an abstraction of a real and conceptually complex system that is designed to display significant features and characteristics of the system, which one wishes to study, predict, modify or control [12]. In our approach, the comprehension process model is a formal description that represents the relevant information resources and their interactions. The information resources include:

- *Task*: description of the comprehension process task.
- *User*: description of the participant who has the main responsibility to fulfill the task.
- *Tools*: description of existing and available program comprehension tools.
- *Artifacts*: description of software and comprehension process artifacts.
- *Software*: description of the target software.
- *Documents*: description of the documented artifacts
- *Historical data*: description of the information collected during the process for later reuse.

In what follows, we describe: (1) the ontological representation used to model the information resources and (2) the story-driven approach used to model the interaction between users and the process context.

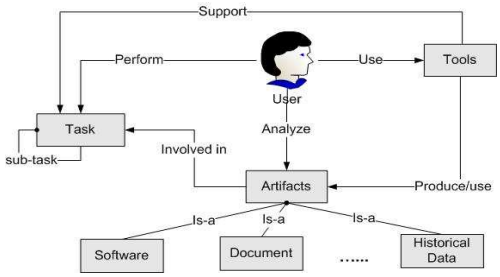

Figure 2 Inter-relationships of the basic elements

4.1 Ontological Comprehension Process Model

In this research, we use ontologies and Description Logics to formally model the major information resources used in program comprehension and their inter-relationships. The benefits of using a DL-based ontology as a means to model the structure of our process model are as follows:

Knowledge aquisition. Program comprehension is a multifaceted and dynamic activity involving different resources, which utilizes these resources to enhance the current knowledge about a system. Consequently, any comprehension process model has to reflect and model both the knowledge acquisition and use of the newly gained knowledge. The ontological representation provides the ability to add newly learned concepts and relationships as well as new instances of these to the ontological representation. This extendibility enables our process model to be constructed in an incremental way, closely modeling the same iterative knowledge acquisition behavior used to create a mental model as part of human cognition of a software system [1-4].

Reasoning. Having DL as a specification language for a formal ontology enables the use of reasoning services provided by DL-based knowledge representation systems. Furthermore, reasoning services provide the ability for inferring knowledge through transitive closure across different ontologies. Using such a DL-based ontology representation and its reasoning capabilities provides an active guidance and support during knowledge acquisition.

Building a formal ontology for program comprehension requires an analysis of the concepts and relations of the discourse domain. In particular, the process model outlined above must be supported by the structure and content of the ontological knowledge base. Our approach here is twofold: We (1) created sub-ontologies for each of the discourse domains, like tasks, source code, software documentation, tools and stories (Figure 2); and (2) linked them via a number of shared high-level concepts, like *process, artifact,* or *task,* which have been modeled akin to a (simple) upper level ontology [9].

Table 2 Software comprehension ontology (partial view)

Concept	Definition	Example Instances
Software sub-ontology		
Software	Source code model	Source code model of uDig system [18]
Class	Class of the system	AddessSeeker, USGLocation
Task sub-ontology		
Architecture analysis	Analysis of major components and their inter-relationships	Architecture analysis of uDig system
Tool sub-ontology		
Metrics	Source code analysis and measurements	JDepende4Eclipse, CodeCrawler, SOUND
Class analysis	Analysis of class models	Creole, SOUND
Coupling analysis	Analysis of component relationships	Creole, SOUND
Story sub-ontology		
Protagonist	The analyzer	Concordia SEGroup
Document sub-ontology		
Javadocs	Javadocs for the system	uDig Javadocs
User guide	The user's guide	uDig help/infocenter

Each sub-ontology contains concepts (TBox) and relations particular to its domain. For example, the source code ontology holds information about source code entities, like classes, methods, or variables, and their relations. Instances (ABox) are provided automatically by analysis tools, like our source code ana-

0-7695-2698-5/06 $20.00

lyzer [21], modeled manually, like tool instance information, or supplied by a user during the comprehension process. The various connections between the sub-ontologies, through the upper level as well as additional shared concepts, allow us to *reason* across ontology boundaries, both on concepts and instances. For example, links between the source code and documentation ontologies allow us to automatically analyze both correspondences and inconsistencies between source code and its documentation using our automated reasoner, *Racer*. Some ontology concepts and instance examples are shown in Table 2. The implementation of our ontology will be further discussed in Section 5.

4.2 A Story-Driven Interaction Model

As mentioned previously, one essential aspect of a comprehension process is to provide for an interaction between users and the comprehension process in the context of a current comprehension task. In an interactive comprehension story, the user is immersed in the program comprehension environment in which a particular comprehension task (story) unfolds and the user is considered as an active character in the story, able to interact with different elements (various resources (e.g. supports from tools, techniques, documents and expertise etc.) and other characters (e.g. system or historic user data) to perform a particular comprehension task.

Table 3 Mapping between Story and Comprehension

Story Model		Comprehension Model
Theme		Task goal
Genre		Cognitive model: top-down, bottom-up, user-defined integrated approach
Characters	Protagonist	The current user who perform the task
	Impact/other characters	Historical user data, that is related to the current task, wizard/system
Settings		Program comprehension resources: software, tools, techniques, etc.
Plot		Process management
Throughline		Interactive communication between user and plot
Interaction		Inter-relationships between the characters and environment resources
Conflict		Shortage of resources, unsuccessful use of a tool, technique
Climax/Resolution		Outcome of the process (success/failure)

From a story perspective, the comprehension process can be viewed as authoring an interactive narrative between users and systems towards completing a specific goal. The similarities among the story model and program comprehension process model allow that both can be mapped to the formal representation (using ontologies and DL). The story model represents interactions among users and the process. Both the story interaction and the triggering of events and actions during the comprehension process are managed by the story model and its manager. One of the advantages of the story metaphor is that it provides an interaction context to guide the comprehension process. Table 3 illustrates this mapping between the comprehension and story model.

A typical tool usage scenario for our comprehension process model is illustrated in Figure 3, with the iterative nature of the process being reflected by the loop (messages 2-22). A user is the protagonist in the story who has a comprehension task at hand to be completed. The story manager, ontology manager and reasoner are all impacted characters providing assistance to the user. Resources are the available elements in the story environment, and users can use them during the task solving process. The *story manager* is the main character with whom the user interacts during the task solving process.

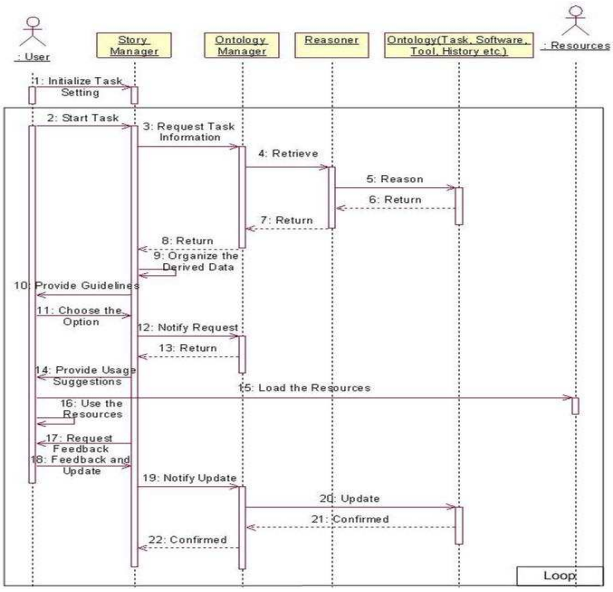

Figure 3 Tool Usage Sequence Diagram

Based on a given task setting (message 1), the story manager then applies a set of predefined queries to identify potential tools and techniques that might be suitable to comprehension tasks (message 2-14). The user interacts with the process through the story metaphor, providing the current comprehension context and the ability to trigger new events and actions. One of the major advantages of our ontological representation is not only the ability to reason across the different information resources, but also the ability for users to add newly gained knowledge (concepts, relations and instances) to the system, and make these available for further processing. For instance, message 9 may return a list of techniques that support architecture analysis such as class model recovery, design pattern analysis, coupling analysis and metrics. The resulting set of tools

will be further analyzed and a potential applicability ranking is derived. At this point, the protagonist has the choice between three different options. (1) Accept one of the suggestions (shown in Figure 3 – messages 15-18), or (2) explore all tools and techniques that are registered with the process model, as well as other potentially relevant information stored in the ontologies. (3) Create their own queries to search and filter information for specific settings, tools or historical data.

After completion of the tool usage, the protagonist will provide feedback and annotate briefly their experience with the tool and their success towards problem solving (Messages 18-22). The resulting feedback is used to enrich the historical data ontology, as well as triggering the next iterations of the comprehension process (depending on the outcome of the current step success or conflict/failure).

5. System Implementation and Evaluation

In this section, we provide a more detailed overview of our system implementation followed by a brief discussion on the evaluation of the process model.

5.1 Process Implementation Details

A general system overview for our process model is shown in Figure 4. The process itself is supported by two main components, the ontology manager and its query Interface [21] and the story manager.

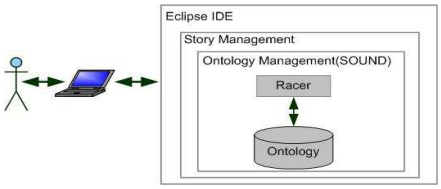

Figure 4 System Architecture

Ontology management is used to manage the infrastructure of the process model, where the basic elements of the program comprehension process and their inter-relationships are formally represented by DL-based ontology. It supports the creation of new concepts and their relations in any of the given sub-ontologies, coordinates the reasoner with the ontologies, provides and controls querying and reasoning services across the different sub-ontologies. The user is allowed to perform both pre-defined queries and user-defined queries managed by the ontology management. The ontology manager is based on our SOUND (Software Ontology for UNDerstanding) tool, an Eclipse Plug-in that provides both ontology management (software ontology and document ontology have been developed) and inference service integration using the Racer [22] reasoner. So far, the ontology management interface provides these services: adding/defining new concepts/relationships, specifying instances, browsing the ontology, and queries/reasoning.

Story management is built on top of the ontology manager infrastructure. The story manager provides for the user both the context and the interactive guidance during the comprehension process. The story manager coordinates the protagonist (the user) to dynamically author the specific comprehension task solving story. In addition, the story management is also responsible for providing an intuitive visual metaphor to vividly model the whole task solving process. During the process, the story manager is monitoring events and triggers that might affect the storyline of the comprehension process, e.g. user or system activities.

5.2 Evaluation

At the current stage, we have successfully implemented and used our SOUND ontology management and query tool to perform comprehension tasks such as impact analysis, design pattern recovery, and component identification [3, 21]. In addition, we have defined an initial set of concepts and relations for the remaining sub-ontologies as the foundation for our process model. A set of frequently used queries has been defined in the system, e.g. identifying the coupling among classes, recovering the design pattern in the system, deriving measurements from the system, etc. [3]. We are now in the process of conducting a larger case study in collaboration with Defence Research and Development Canada (DRDC) - Valcartier to explore and validate the applicability of our comprehension process model. The task to be performed relates to component substitutions, which can be seen as a specific instance of a software evolving task. This case study uses a Geographic Information System (GIS) – uDig [18]. The substitution task involves the identification of the tracking component and replacing it with a new one meeting the client's requirements. User profiles will be set up to track the progress of users (with different expertise levels) in completing the substitution task. Feedback from the process and information resource usage will be collected for further refinement of the process.

6. Discussion and Related Work

Existing work related to software maintenance processes [19, 25, 29] typically are very general and lack formal representations. Thus, they are unable to utilize any type of reasoning services across the different

knowledge sources involved in the comprehension process. To the best of our knowledge, there exists no previous work that focuses on developing a formal process model to describe the program comprehension process and to allow maintainers to interact through the story driven model with the process itself.

Existing work on comprehension tool integration focuses either on data interoperability using a common data exchange format [25] or on service integration among different reverse and software comprehension tools [20]. Our approach can be seen complementary to these ongoing tool integration efforts. Improving the overall capabilities and applicability of reverse engineering tools will enrich our tool ontology, benefiting our comprehension process model. However, our approach goes beyond just tool integration, by providing a formal ontological representation that supports both reasoning across different knowledge sources (including tools) and provides context support and guidance during the comprehension process itself.

7. Conclusions and Future Work

Program comprehension is an essential part of software evolution as any software evolving activity cannot skip the first comprehending step. Our work promotes the use of both formal ontology and automated reasoning in program comprehension research, by providing a DL-based formal ontological representation of different information resources. The flexibility and extensibility of the ontologies also support the evolution of our comprehension process model. The story model provides for the interaction and context visualization between the process model and the user. The intuitive visual format for the task story model is still in development. As part of our future work, we will conduct several case studies to enrich our current ontologies and optimize the comprehension process model for different comprehension and software evolution tasks.

Acknowledgement:
This research was partially funded by DRDC Valcartier (contract no. W7701-052936/001/QCL).

References

[1] B. Shneiderman, “Software Psychology: Human Factors in Computer and Information Systems”. Winthrop Pub.1980.

[2] R. Brooks, “Towards a theory of the comprehension of computer programs”.Int. J. of Man-Machine Studies, pp. 543-554, 1963.

[3] Y. Zhang, R. Witte, J. Rilling, V. Haarslev, “Missing Links–An Ontology-based approach to Traceability among Software Artifacts”, *ACM SIGDOC’06. Submitted.*

[4] S. Letovsky, “Cognitive processes in program comprehension”. In Empirical Studies of Progr. pp. 58-79, Ablex Pub. 1986.

[5] M.A. Phillips and C. Huntley, “Dramatic A New Theory of Story”, 5th Edition, Screenplay Systems Inc., 2001.

[6] J. Bruner “Acts of Meaning”, Cambridge, MA: Harvard University Press, 1990.

[7] M. O. Riedl, R. M. Young. “An Intent-Driven Planner for Multi-Agent Story Generation”, Proc. of the 3rd Int. Conference on Autonomous Agents and Multi Agents Systems, 2004.

[8] A.V.Mayhauser, A.M.Vans, “Program Comprehension During Software Maintenance and Evolution”, IEEE Computer, pp. 44-55, Aug.1995.

[9] Niles and A. Pease. “Towards a Standard Upper Ontology”. In C. Welty and B. Smith, editors, Proc. of the 2nd Int. Conf. on Formal Ontology in Information System (FOIS), Maine, 2001.

[10] H.Muller, J.Jahnke, D.Smith, M.-A.Story, S. Tilley, and K. Wong, “Reverse Engineering: A Roadmap”, The Future of Software Engineering, ACM Press 2000.

[11] C.Riva, "Reverse Architecting: An Industrial Experience Report", 7th IEEE WCRE, pp. 42-52, 2000.

[12] M. I.Keller, R. J. Madachy, and D. M.Raffo, “Software Process Simulation Modeling: Why? What? How?”. Journal of Systems and Software, Vol.46, No.2/3, 1999.

[13] M.-A.Storey, “Theories, Methods and Tools in Program Comprehension: Past, Present, and Future”, 13th IWPC 2005 pp.181-191.

[14] P.Kroll, P.Kruchten,“The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP”, AW, 2003.

[15] N. Gershon, W.Page, “What storytelling can do for information visualization”, CACM 8(44), pp.31-37, 2001.

[16] M.J.Pacione, M.Roper, M.Wood, "A Novel Software Visualisation Model to Support Software Comprehension", 11th WCRE, pp.70-79, 2004.

[17] E.S.Hartland, “The science of fairy tales”. Walter Scott, 1891.

[19] IEEE Standard for Software Maintenance, IEEE 1219-1998.

[18] http://udig.refractions.net/confluence/display/UDIG/Home, accessed 6/2006.

[20] D.Jin and J.R.Cordy. "Ontology-Based Software Analysis and Reengineering Tool Integration: The OASIS Service-Sharing Methodology". 21st IEEE ICSM 2005, Budapest, Hungary, September, 2005.

[21] Y.G.Zhang, J.Rilling, V.Haarslev, “An ontology based approach to software comprehension – Reasoning about security concerns in source code”. In Proc. of 30th Int. Computer Software and Applications Conference, 2006.

[22] V.Haarslev and R.Moller, “RACER System Description”, In Proc. of Int. Joint Conference On Automated Reasoning (IJCAR’2001), pp.701-705, June 2001.

[23] M.O.Riedl, R.M.Yong, “From Linear Story Generation to Branching Story Graphs”, IEEE Computer Graphics and Applications, IEEE Computer Society, 2006.

[24] OWL Web Ontology Language Reference, W3C Recommendation, http://www.w3.org/TR/owl-ref, accessed June 2006.

[25] M.-A. D.Storey, Susan Elliott Sim, Kenny Wong, “A collaborative demonstration of reverse engineering tools”, ACM SIGAPP Applied Computing Review, Vol. 10(1), pp18-25, 2002.

[26] P.Wongthongtham, E.Chang, T.S. Dillon, “Towards “Ontology”-based Software Engineering for Multi-site Software Development”, 2005 3rd IEEE Int. Conference on Industrial Informatics (INDIN), 2005.

[27] P.N. Johnson-Laird, “Mental models: towards a cognitive science of language, inference and consciousness”, Cambridge, Mass. Harvard University, 1983.

[28] F.Baader, D. Calvanese, D. McGuinness, D.Nardi, P.P.-Schneider, “The Description Logic Handbook”, Cambridge University Press, 2003.

[29] KH.Bennett, VT.Rajlich, “Software maintenance and evolution: a roadmap”. Proc. of the Conference on The Future of Software, pp.73-87, 2000.