MITRE

Reverse Engineering Cognition

Maura K. Tennor
Cyber Operations Research, Lead, J52B
Internal Research and Development Portfolio

July 2015

Approved for Public Release;
Distribution Unlimited. 15-2630

Ft. Meade, MD

Abstract

This paper presents the results of a literature review on the topic of Reverse Engineering Cognition under MITRE's Internal Research and Development Portfolio. Resource material for this review was gathered by conducting a series of searches for journal articles, conference proceedings and a variety of Internet sources. The paper summarizes what we know today about how reverse engineering of binaries is performed at a cognitive and mechanical level, ties in related areas such as expertise and mental models, and suggests avenues for future research.

Table of Contents

List of Figures

1 Introduction

Reverse engineering (RE) skills are in high demand. Most organizations employ a journeyman approach to training new reverse engineers, in which novices work side-by-side with those more experienced in the field until they are able to work independently. The experts guide the novices on a problem, sharing experiential knowledge and other tricks of the trade. This requires substantial time investment on the part of both experts and novices and takes the experts away from performing reverse engineering tasks themselves to perform this mentoring function. While this method is effective in transferring skills and knowledge to newcomers, it is time consuming and inefficient. We would like to discover ways to shorten the time it takes to become an expert in reverse engineering, as well as ways to make the development of new reverse engineers less dependent on the time of current experts.

The state of the art for tools that support RE requires advancement as well. Due to the high demand for RE, those engaged in the activity can use every assistance in making their work more efficient. While great strides have been made in automation (for malware analysis, for example), there will still be a need for manual RE for the foreseeable future. With that in mind, we would like to discover enhancements to the RE toolset that could better support the cognitive and mechanical processes of RE.

Therefore, the Cyber Effects Portfolio posed the following research questions:

- Can reverse engineers articulate how they construct their mental model for a piece of software?
- Can the mental models be articulated, codified, and can the written form be used by others to improve their skills at reverse engineering?
- Can additional tools be developed to better support the reverse engineering process?

This study seeks to codify the process of reverse engineering in order to begin reducing the time required to master the art, and to make knowledge transfer more scalable. The study began with a comprehensive literature review to better understand the problem space and prior work that might help address the research questions. This paper summarizes the results of the literature review and proposes a research agenda.

2 Literature Review

Resource material for this review was gathered by conducting a series of searches for journal articles, conference proceedings and a variety of Internet sources. Section 5 contains a full list of the references that were reviewed.

2.1 What is Reverse Engineering?

Reverse engineering is "the process of extracting the knowledge or design blueprints from anything man-made…Reverse engineering is usually conducted to obtain missing knowledge, ideas, and design philosophy when such information is unavailable…" (Eilam, 2005, p. 3). The term can apply to anything from a chemical substance to a physical object to software code that one wishes to decompose in order to discover how it was made. "Software reverse engineering is about opening up a program's 'box,' and looking inside… [It is] a purely virtual process, involving only a CPU, and the human mind" (Eilam, 2005, p. 4).

The primary goals of software RE are to discover general properties of the program such that one can construct a complete "picture" of the program. This involves understanding how the program uses the system interface, understanding the program's functions and instruction-level information, and understanding how the program uses data (Bryant, 2012).

2.2 Reverse Engineering for Software Maintenance

A broad body of work exists that investigates reverse engineering for the purpose of software maintenance (and the tools that support it). As code bases grew, the need for RE skills and technologies grew as well. It became increasingly likely that developers would have to maintain or update code they had not originally developed, with a spike in demand for RE emerging around the Y2K problem at the turn of the millennium (Storey, 2005). The software maintenance RE community established an annual IEEE conference called Software Analysis, Evolution and Reengineering in 1993[1].

While RE of binaries does not afford natural language cues found in source code such as function names or comments, there are still commonalities with RE for software maintenance. For example, the underlying goal is to discover the human-oriented concepts from the code. Additionally, domain knowledge can be quite helpful (i.e., it helps to understand the ticketing process when reverse engineering ticketing software).

Of particular importance to the research questions posed for this study, the RE for software maintenance work identifies differences between novices and experts. (Soloway & Ehrich, 1984) established that expert programmers exhibit two kinds of knowledge that novices do not, namely programming plans (i.e., typical sequences of actions in programs) and rules of programming discourse. Additional research shows that experts chunk better (Gobet, 2005), and experts use a breadth-first approach, and adopt a system-wide view, whereas novices use both breadth and depth-first approaches and do not think

[1] For more detailed information and history, see:
http://saner.soccerlab.polymtl.ca/doku.php?id=en:past_editions and
http://saner.soccerlab.polymtl.ca/doku.php?id=en:start

in system terms. Further, experts make more use of external devices as memory aids, experts tend to reason about programs according to functional and object oriented relationships, while novices tend to focus on objects (Storey, 2005). Validating these differences between novice and expert RE's of binaries would be a valuable exercise, and more efficient that repeating similar studies with a slightly different population.

The RE for software maintenance community also made great strides in examining requirements for proper tool support of RE. For example, (Tilley, 1998) developed a framework for RE environments that supports the cognitive processes of RE as reflected in a hierarchy of tasks and activities required for RE. (Storey, 2005) outlined several requirements for RE tools such as context driven views, browsing and search capabilities, and scratch pads. Again, validating these requirements for binary RE could expedite binary RE tool development.

2.3 Cognitive Processes, Mental Models & Sensemaking

The RE for software maintenance community paved the way for understanding RE from a cognitive perspective. For example, (Biggerstaff, Mitbander, & Webster, 1993) established that a person understands a program when they can declare the computational intent in human oriented terms. More specifically,

> "A person understands a program when they are able to explain the program, its structure, its behavior, its effects on its operational context, and its relationships to its application domain in terms that are qualitatively different from the tokens used to construct the source code of the program" (Biggerstaff, Mitbander, & Webster, 1993, p. 482).

Similarly, (Rajlich, 2009) described RE as a matter of *recovering a programmer's intentions*. (Byrne, 1992) presented a conceptual foundation for RE involving levels of abstraction from implementation to conceptual, wherein each level of abstraction describes a different aspect of the system.

(Erdos & Sneed, 1998) outlined the following seven questions for programmer to maintain program they only partially understand:

1. "Where is a particular subroutine/procedure invoked?
2. What are the arguments and results of a function?
3. How does control flow reach a particular location?
4. Where is a particular variable set, used or queried?
5. Where is a particular variable declared?
6. Where is a particular data object accessed?
7. What are the inputs and outputs of a module?" (Erdos & Sneed, 1998, p. 99).

Some of the research on cognition in reverse engineering aims to inform automation of RE, and hence ventures into artificial intelligence territory. At present, automation of RE is limited by the extent to which RE involves assigning meaning, as this activity is not algorithmic (Bryant, 2008). Similarly, (Heelan, 2011) posits that cyber defense research often fails because it attempts complete automation, which isn't feasible in real life. Rather, he argues, automation should be integrated into a human's workflow.

2.4 Expertise

In addition to the research on expertise specific to RE for software maintenance mentioned above, research on expertise in general proves valuable to our inquiry as well. Much research has been devoted to investigating the minimum period required to attain expert levels of performance in any field (Ericsson & Charness, 1994). These efforts have converged on a 10-year-rule – or, as popularized in Malcom Gladwell's "Outliers", a 10,000 hour rule (Gladwell, 2008).

Furthermore, research has shown that those hours or years are best spent in *deliberate practice,* which is defined as specialized training tasks chosen by a qualified instructor, "motivated by the goal of improving performance" (Ericsson & Charness, 1994, p. 738). In other words, playing at the activity, or even engaging in it for social or monetary reward (as work) will not produce the same results (Ericsson & Charness, 1994).

This concept was further developed as articulated here:

> "…accelerated acquisition of domain experience and the development of intuition-related skills can be facilitated through deliberate practice, critical self-appraisal, and candid feedback… through implicit learning, situated training, deliberate practice, self-critique, and metacognitive instruction, warfighters could enhance their intuition-informed situation assessments, and gain regulatory skills to more deliberately control their intuitive processing" (Bartlett, Nolan, & Marrafino, 2013, p. 5).

Due to the large amount of knowledge required to begin comprehending RE, undergraduate programs typically do not cover the subject. Current IEEE standards for undergraduate Computer Science (The Joint Task Force on Computing Curricula, 2013) and Software Engineering (The Joint Task Force on Computing Curricula, 2004) place reverse engineering as a minor subtopic under software maintenance. Even at the graduate level, it is unlikely for a student to encounter much instruction on RE (Stevens Institute of Technology, 2009). Therefore, much of the training and education in RE takes place either on one's own time or in the workplace.

At present, in many environments where RE takes place, the operations tempo in those environments necessitates that people learn on the job (OJT). By default, then, the practice they engage in comprises whatever the problems "du jour" are, rather than an ordered sequence of specialized tasks that help them build knowledge in a deliberate manner.

As stated above, much RE training follows a journeyman approach, in which novices work side-by-side with those more experienced in the field until they are able to work independently. (Tucker, 2003) states that imposing an expert's cognitive model on a novice doesn't always work, cautioning that a student can get "confused in trying to understand a view of the abstraction to which they do not personally relate" (Tucker, 2003, p. 3). (Tucker, 2003) further cautions against emphasizing products over process with students. This raises particular concerns with OJT, as customer requirements can necessitate emphasis on products over process.

(Cowley, 2014) identified five stages/milestones in a reverse engineer's progression from novice to expert:

1. **Novice:** Tool proficiency (IDA Pro, compilers, debuggers, etc.)
2. **Apprentice:** Novice becomes operational, indicated by significant reduction in assistance required to execute daily work (about 8-12 months in)
3. **Journeyman:** Junior RE's knowledge is comparable to the expert's – junior RE attempts same strategies as expert but without expert's guidance
4. **Expert:** Organizational recognition/promotion to senior RE job (5-7 years in)
5. **Master:** Solves many problems without assistance of another engineer. The RE can "persist through his or her own ignorance" (Cowley, 2014, p. 11).

Note that in some cases, these serve more as indicators of a reverse engineer's growth in expertise than as descriptions of the proficiency itself.

(Cowley, 2014) summarizes expertise in RE in the following manner:

> "Expertise in this domain does not necessarily entail the development of detailed knowledge and skills on a few topics; it is about developing sufficient breadth of skills and knowledge on computer software and hardware to enable finding the information needed to understand how the malware impacts the hardware, software, and respective network. In addition, knowledge built from repeated experiences analyzing malicious code enables expeditious problem solving, which is especially important for time-sensitive, sponsor-driven work" (Cowley, 2014, p. 18).

2.5 RE of Binaries

This section covers emerging research specifically focused on the cognitive aspects of reverse engineering of binaries. First, we will illustrate the mechanical process of RE and why it is hard to do. Next, we will describe the findings of two key papers in this area.

2.5.1 Mechanics of RE

(Pozluszny, 2015) presents a generic process for RE, as follows:

1. "Gather information
   - IAT (imports table)
   - strings
   - dynamic analysis
2. Identify function of interest
3. Identify CALLs
4. Identify algorithms and data structures
5. Pseudo-code it!
   - if having trouble, draw the memory and CPU and map what happens at each instruction
6. Rename function(s), argument(s), variable(s)
7. Add comments
8. GOTO 2" (Pozluszny, 2015)

In the process of reverse engineering, analysts examine some or all of the following (excerpted from (Bryant, 2012) and abbreviated):

- **assembly instructions** – instructions are usually data movement, arithmetic or control-flow operations
- **program data** – have to figure out how the data is used and what data is used by which instructions in the program. Other data of interest:
    - processor's register values
    - program's stack memory values
    - processor's flag values
    - file or registry contents the program accesses and/or changes
- **control flow info** – allows you to infer causality between instructions in a program
- **functional info** – breaks programs into subroutines, instruction sequences and blocks demarcated by control flow instructions
- **other observable features** – observations about how software interacts with the operating system and hardware, things that can be seen while a program is running (a window opening, files being created, etc.), how much the processor is being used at a given time (Bryant, 2012, pp. 41-45).

2.5.2 Why RE is Hard

Many factors contribute to making RE is so difficult to perform. To begin with, one must master a lot of knowledge to even get started. The list of items reverse engineers must examine outlined in the previous section illustrates some of the knowledge required. Other specialized knowledge might include:

- Translating from assembly into higher-level languages
- Specific hardware architecture issues
- OS API functionality
- OS internals knowledge
- How compilers generate machine code
- Classes of vulnerabilities and other weaknesses (for malware analysis)
- Knowledge of the type of software being analyzed
    - Examples for malware – working knowledge of software protection techniques and how they work, crypto algorithms, …
    - Examples for kernel code - knowledge about re-entrant code, critical code sections, interrupts, dead lock issues, … (Bryant, 2012).

In addition, reverse engineers must use specialized tools such as disassemblers, hex editors, and decompilers (Bryant, 2012). (Song, et al., 2008) articulated several other reasons RE is hard to do, to include complexity, lack of higher-level semantics and

obfuscation. Further, assembly code is different for each architecture so one starts from scratch on that aspect each time one encounters a new architecture. In some cases, having to achieve a whole system view of the code poses a challenge as well (Song, et al., 2008).

(Zayour & Lethbridge, 2000) assert that RE entails bearing a heavy cognitive load, which they define as "the amount of attention and mental energy required to accomplish a task…amount of cognitive resources required…and load placed on short-term memory, which represents the human central processor…" (Zayour & Lethbridge, 2000, p. 2). Cognitive load includes aspects such as the level of enjoyment received from performing a task, the likelihood of making errors, and the energy required to complete the task.

Indeed, the current state of RE tools necessitates that reverse engineers organize and maintain a lot of information in their heads or that they track it manually. MITRE's internal Intro to RE course advises the following:

> "I **strongly recommend** that while you're looking the code over, you do one or more of the following:
>
> - Write pseudo code (or real code, whatever you prefer)
> - Draw memory / the stack to help you see what a program's doing
> - Rename functions, variables, addresses
> - Comment things as you figure out what they do
> - Keep a list of "what do I know so far" when you have a specific goal in mind" (Clapis, 2015).

(Clapis, 2015) also recommends keeping track of the stack contents (manually) in one's notes via a chart such as the following:

```
[       ] 20
[       ] 1C
[       ] 18
[       ] 14
[       ] 10
[       ] C
[       ] 8
[       ] 4
[       ] ebp (base)
[       ]
[       ] userInput
```

Aside from renaming functions, etc. and commenting, these actions are carried out in a simple notepad or other generic tool not specifically designed to capture one's mental model of a program. By way of further illustration, consider the following screen shots from the commonly used RE tool IDA Pro. Figure 1 shows a piece of code in IDA's "text" view.

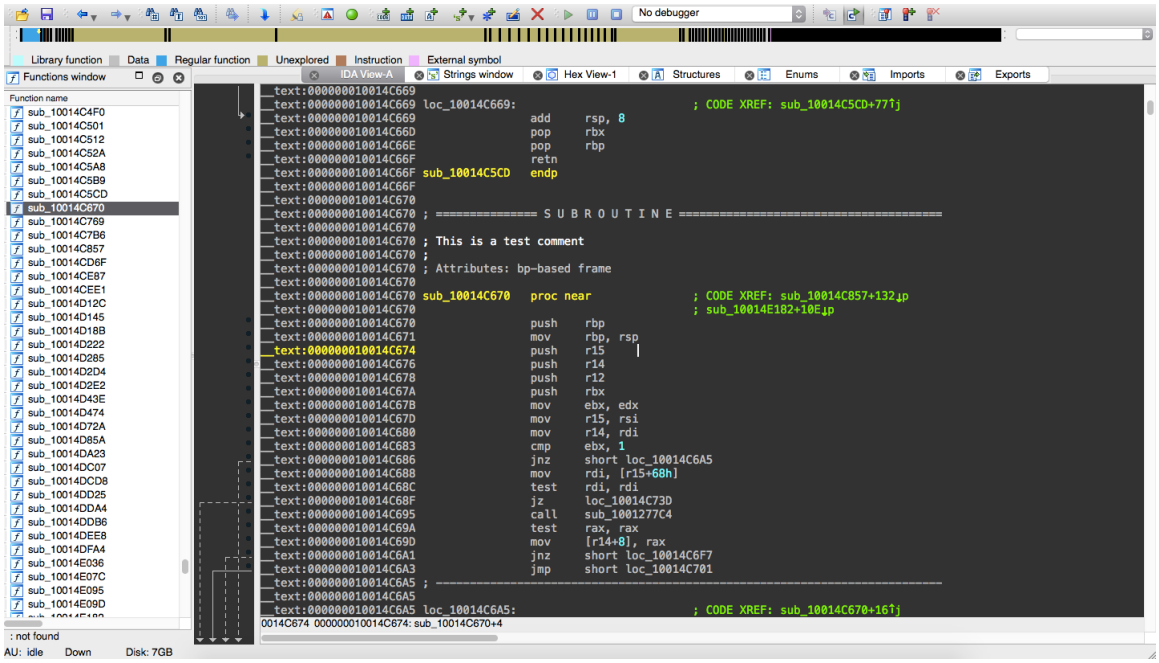

Figure 1

Alternately, one can view the code in IDA's "graph" mode, as shown in Figure 2. "This very handy feature helps to isolate code into basic blocks with edges that identify the kind of code flow between blocks. These graphs can help identify conditions, loops, call graph and exit blocks" (Posluszny, 2015).

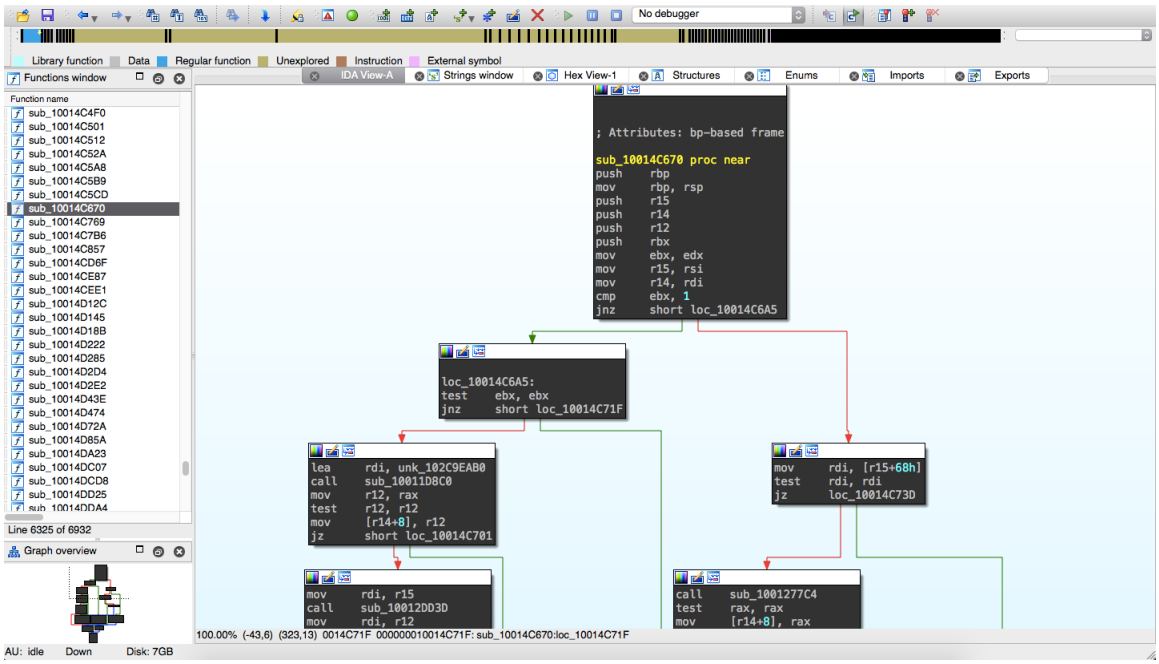

Figure 2

Figure 3 illustrates an example of a comment added to the code (see the red arrow).

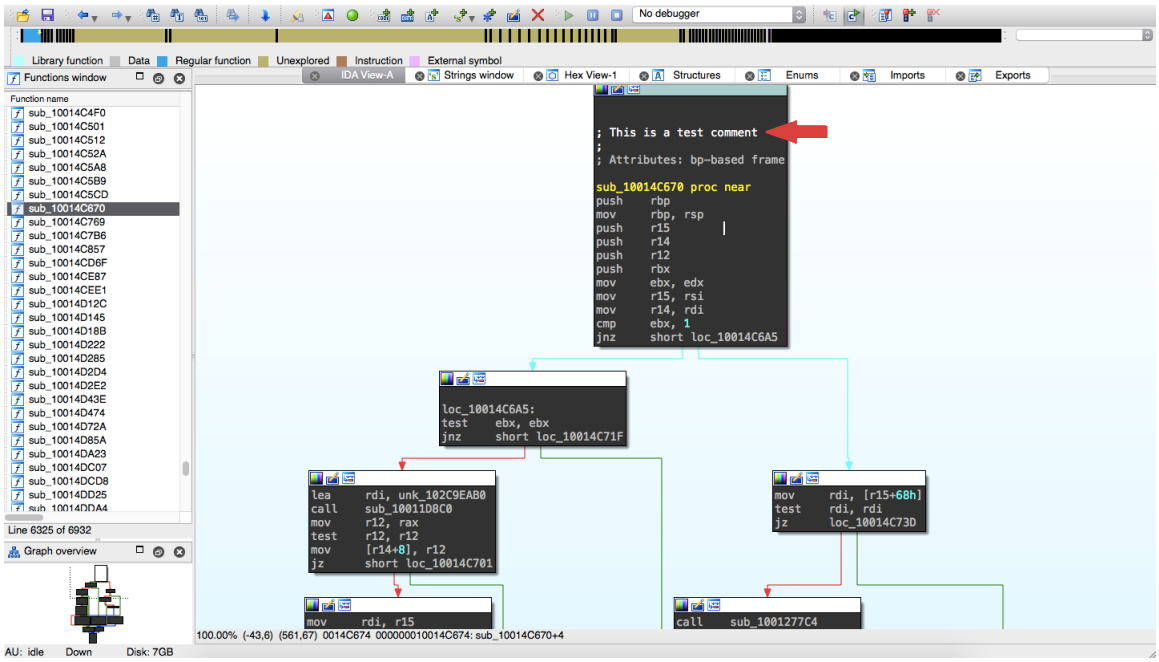

Figure 3

2.5.3 RE as Sensemaking

Much like the process of abstraction described by (Byrne, 1992), (Bryant, 2012) demonstrated that binary RE is an iterative process of abstraction from low level data to high level concepts, connecting input from the environment with existing mental model, including things like control flow representations and/or programming plans.

Regarding the research questions on reverse engineers' mental models, (Endsley & Models, 2000) proffer that although software developers are taught to design software that supports the user's mental model, it can be extremely difficult to obtain an objective, complete understanding of what the mental model is. They further illustrate the quandary that the existence of mental models is equally difficult to prove as their non-existence.

Thus, (Bryant, 2012) describes RE as a process of sensemaking, akin to the development of situational awareness, in which one reasons and forages through information to construct a mental model. In his assessment, the model is dynamic, continuously updated via an iterative loop of identifying knowledge gaps, seeking information and/or structure for information, and then adding, tuning and adjusting the mental model. Figure 4 below illustrates Bryant's sensemaking RE process (Bryant, 2012, p. 169).

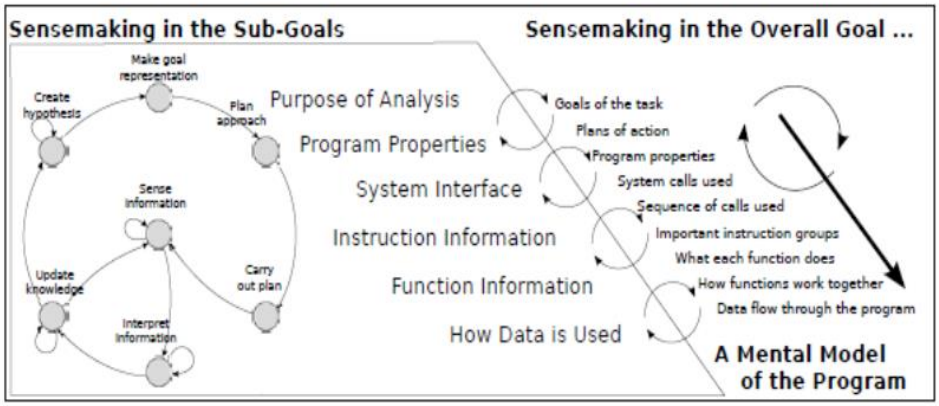

Figure 4

2.5.4 Requirements for Successful RE

Based on a case study of a team of malware analysts, (Cowley, 2014) identified potential requirements for successful RE. She took a holistic approach, examining both cognitive and non-cognitive aspects of individuals performing the task, in the context of the team and organization in which the individuals operate. Some of the **cognitive** abilities[2] she identified as potential requirements were:

1. Category flexibility – ability to generate or use different sets of rules for combining or grouping things in different ways
2. Deductive reasoning – ability to apply general rules to specific problems to produce answers that make sense
3. Flexibility of closure – ability to identify or detect a known pattern (a figure, object, word or sound) that is hidden in other distracting material
4. Fluency of ideas – ability to come up with a number of ideas about a topic (number of ideas is important, not quality, correctness, or creativity)
5. Inductive reasoning – ability to combine pieces of information to form general rules or conclusions (includes finding a relationship among seemingly unrelated events)

(Cowley, 2014) categorized the non-cognitive potential requirements as pertaining to novices, experts, teams and organizations. Table 1Table 1 below lists some of the **non-cognitive** potential requirements she identified.

Novice	**Expert**
• Interest in solving problems • Self-motivated • Conscientious • Takes initiative • Shows creativity	• Persistence • Passion for the work • Openness to experience • Minimally intimidated by problem solving

[2] See report for full listing.

	• Self-motivated • Humility
Team • Appropriate work pacing • Provide appropriate training • Enable autonomy	**Organizational** • Work/life balance • Support development of expertise • Provide appropriate tools

Table 1

3 Addressing the Research Questions

Recall that the research questions posed were:

- Can reverse engineers articulate how they construct their mental model for a piece of software?
- Can the mental models be articulated, codified, and can the written form be used by others to improve their skills at reverse engineering?
- Can additional tools be developed to better support the reverse engineering process?

Recall that (Endsley & Models, 2000) caution that the pursuit of mental models can be misleading. They observe that it can be difficult to obtain an objective, complete understanding of a person's mental model. They even question the existence of mental models. For these reasons, (Bryant, 2012) opted to discuss RE as a situational awareness activity rather than a mental model.

In that light, (Bryant, 2012) covered much of the territory the first two research questions were intended to address. The interviews and direct observation he performed with RE SMEs illustrated that the cognitive process of reverse engineering can be articulated, and that the best description of that process is a sensemaking model. Sections 3.1 and 3.2 of this report discuss how to apply this and other findings from the literature to optimize training. Section 3.3 addresses tool development.

3.1 Closing the Novice-Expert Gap

Firstly, as mentioned in Section 2.2 above, much research has already been done with respect to reverse engineering for the purpose of software maintenance. This research illuminates differences in novices and experts in that context. It might save time and effort to validate these results for novices and experts engaged in binary RE rather than redo the studies with a slightly different population.

Another possibility for future research would be to examine ways to make RE training more effective or efficient, such that the time it takes to develop expertise could be condensed. A common model employed with success by the federal government is a multi-year "internship" or "development program," in which participants engage in professional development via both classroom and on-the-job training[3]. Interns receive mentorship throughout their time in the program, and many such programs allow interns to convert to permanent employees at the conclusion of the program. These programs serve as launch points for the careers of the participants, and in some cases enable the government to vet candidates in real-world positions before making a permanent commitment to them.

[3] http://www.opm.gov/policy-data-oversight/hiring-authorities/students-recent-graduates/#url=graduates

An example of such a development program is the Cryptanalysis Development Program (CADP) at the National Security Agency (NSA)[4]. The CADP is a three-year professional development program, the goal of which is to develop professional cryptanalysts. Much like RE, very little cryptanalysis is taught in undergraduate or even graduate programs because most cryptanalysis happens in classified settings. Participants learn via both formal and on-the-job training, and CADP interns rotate through several different kinds of offices to learn a variety of critical aspects of the work.

Cryptanalysis and RE have a lot in common in terms of the highly detailed computer science knowledge and skills required, the complexity, heavy cognitive load, and the use of specialized tools. Both require mastery of a lot of information before one can even begin to undertake the activities, and both require a high level of frustration tolerance, patience and perseverance. Given these similarities, it is hypothesized that a development program such as the one described above for cryptanalysis professionals would work well for developing RE professionals as well. A possible avenue for future research would be to design a curriculum of coursework combined with a series of operational tours like those of the CADP for the purpose of training reverse engineers.

Organizations with fewer resources than the federal government might be unable to invest in a development program of that magnitude. An alternate research path could be to develop a kind of "lab notebook" consisting of a cumulative sequence of exercises by which one could build RE knowledge and abilities. Some collections of RE training exist, such as: http://opensecuritytraining.info/. These tend to be more of a smattering of topics, loosely categorized as "beginner, intermediate and advanced" levels rather than a logical sequence of exercises that build upon each other.

Both the development program and the lab notebook concepts are in keeping with the concept of directed practice (Ericsson & Charness, 1994). Rather than giving novices whatever problem happens to arise at any given time, these approaches would provide a more organized, structured learning path, which would hopefully make the process more efficient and effective.

Another possible avenue of research would be to design RE coursework around the concept of RE as a sensemaking process put forth by (Bryant, 2012). As noted above, much of the current coursework focuses on the mechanical process of RE. Introducing reverse engineers to the roles of episodic and semantic/conceptual knowledge in addition to procedural knowledge in their work might prove highly beneficial.

3.2 Measuring RE Aptitude and Skill Level

Another question that emerges in the context of this research is how to identify strong reverse engineers: both those without previous experience who have the potential to become strong reverse engineers, and, of those that do have prior experience, which are the most skilled.

[4] https://www.nsa.gov/careers/career_fields/cryptsiganalysis.shtml

As indicated by the IEEE standards for undergraduate Computer Science and Software Engineering programs mentioned above, most job candidates coming straight out of school will have had little to no formal instruction in RE. Candidates might obtain exposure or experience from employment or recreational activities, but anecdotal evidence from the field indicates this is also rare. Therefore, the development of aptitude testing by which employers can identify candidates with strong potential for success could be a valuable pursuit for future research. The National Security Agency employs such testing in recruiting candidates for the CADP discussed above, with much success.

The cognitive and non-cognitive traits of novices and experts identified by (Cowley, 2014) could aid in the development of testing instruments that will precisely and accurately identify the presence of desirable traits. The Center for the Advanced Study of Language (CASL) specializes in language aptitude testing, and has recently ventured into the arena of cyber aptitude as well.[5] Perhaps some of the expertise there could be leveraged to develop aptitude testing specific to RE.

(Goldstein & Hersen, 2000) caution that:

> "…there have been many reports of the failure of objective tests to predict such matters as success in an occupation… objective tests are no longer used to screen astronauts, since they were not successful in predicting who would be successful or unsuccessful… There does, in fact, appear to be a movement within the general public and the profession toward discontinuation of use of personality-assessment procedures for decision-making in employment situations." (Goldstein & Hersen, 2000, p. 8)

However, at the very least, Cowley's findings could aid in the design of interview protocols to improve vetting of job candidates. Another interesting pursuit would be to examine cases in which someone was previously identified as a good candidate for RE work, but proved not to be, in order to discover lessons learned in RE hiring processes.

(Ericsson & Charness, 1994) describe DeGroot's work with identifying ability in chess. He discovered that a person's choice of next move to make in a given chess game demonstrates their level of expertise. The RE research community could pursue the development of a comparable test of expertise, in which the participant chooses the next "move" so to speak in an RE task. Recall, however, that assembly language differs for each computer architecture. Therefore, a first step would be to develop a generic language of RE such that skill could be measured regardless of which computer architecture the engineer is most familiar with. Once a common RE language is in place, or perhaps in parallel, research should pursue the identification of the contextual choices that best highlight expert skill.

[5] http://www.casl.umd.edu/node/2095

3.3 RE Tool Enhancement

As described in Section 2.5.2, the current state of RE tools necessitates that reverse engineers organize and maintain a lot of information in their heads or that they track it manually. Reverse engineers bear a heavy cognitive load, and make use of simplistic tools such as notepads to keep track of what they know about a program. (Miller & DeWitt, 2014) indicate that "…users need a better way to present and record their analysis more efficiently" (Miller & DeWitt, 2014, p. 3). At a more conceptual level, (Bryant, 2012) suggests development of RE tools that implement the sensemaking process he identified to help analysts understand the programs they examine more quickly.

(Miller, Rogers, & Dewitt, 2014) performed a survey of reverse engineers at The MITRE Corporation to document what tools they use as well as highlight improvements necessary in their current toolset. The survey revealed "27 distinct analysis output features" (Miller, Rogers, & Dewitt, 2014, p. 3) and indicated that collaboration support represents the most urgent need for this population. The primary issue underlying the requirement for collaboration support is a need to be able to

> "…enable the sharing of active analysis, searching of historical analysis, and making analysis available for extended analysis can dramatically increase productivity, decrease time of analysis, and can be leveraged as a training tool for onboarding new analyst (sic)" (Miller, Rogers, & Dewitt, 2014, p. 3).

To address concerns raised by the survey, (Miller & DeWitt, 2014) propose development of a distributed, collaborative RE environment, whereby they can begin to "make future analysis more efficient, create new analysis algorithms and processes, and centralize analysis" (Miller & DeWitt, 2014, p. 4). Similarly, (Storey, 2005) suggests developing an Eclipse-like[6] framework for software reverse engineering.

Complementary to the integrated back end proposed by (Miller & DeWitt, 2014), an integrated front-end environment would further enable RE collaboration. The Eclipse-like widgets could be a way to encapsulate and componentize one's mental model about an RE problem and how to solve it. This also supports the (Bryant, 2012) recommendation of "…incorporating conceptual and procedural knowledge into reverse engineering tools" (Bryant, 2012, p. 172). Widgets can also be reused for future analysis, making the process more efficient.

The RE community should work toward common ontologies and vocabularies to further enable collaboration and reuse. For example, the Malware Attribute Enumeration and Characterization (MAEC[7]) language is a structured, common markup language by which to communicate malware information. To the extent possible, a similar but more generic concept should be developed for all RE as additional support for addressing the collaboration issue.

[6] https://eclipse.org/
[7] http://maec.mitre.org/

The RE for software maintenance community has contributed a large body of research regarding tool support for RE that could possibly be leveraged or at least inform tool research for RE of binaries. (Storey, 2005) provides a thorough summary of this research and captures a list of functions that RE tools need to be able to support. These functions include browsing, searching, and context-driven views (inheritance hierarchy for object oriented code, for example, or call graph view for flat inheritance hierarchy). As illustrated in the IDAPro screen captures (Figure 1, Figure 2 and Figure 3) above, visualization and flexibility thereof appears to be an area ripe for enhancement.

4 Summary/Conclusions

Reverse engineering (RE) skills are in high demand. Learning to expertly reverse engineer binary files requires a lot of time and a specific collection of cognitive and non-cognitive traits. We would like to discover ways to shorten the time it takes to develop expertise in reverse engineering, as well as ways to make expertise development less dependent on the time of current experts. The state of the art for tools that support RE requires advancement as well.

The literature review shows the cognitive process of reverse engineering has been well described by (Bryant, 2012) as an iterative, dynamic, sensemaking model. (Cowley, 2014) articulated cognitive and non-cognitive requirements for successful RE at individual, team and organizational levels. These and other key findings illuminate ways to improve the process of developing expertise by aiding in the identification of high-potential candidates, suggesting how we might better train reverse engineers, and informing RE tool design. This report recommends avenues for future research in each of those domains.

5 Bibliography

Bartlett, K., Nolan, M., & Marrafino, A. (2013). Intuitive Sensemaking: From Theory to Simulation Based Training. In D. Schmorrow, & C. Fidopiastis, *ACH/HCII 2013* (pp. 3-10). Berlin Heidelberg: Springer-Verlag.

Biggerstaff, T., Mitbander, B., & Webster, D. (1993). The concept assignment problem in program understanding. *Proceedings Working Conference on Reverse Engineering* (pp. 27-43). IEEE.

Bryant, A. (2008). Toward Detecting Novel Software Attacks by Using Constructs from Human Cognition. *The 3rd International Conference on Information Warfare and Security* (pp. 59-66). Omaha, NE: Academic Publishing Limited.

Bryant, A. (2012). *Understanding How Reverse Engineers make Sense of Programs from Assembly Language Representations.* Wright Patterson Air Force Base, Ohio: Department of the Air Force, Air Force Institute of Technology.

Byrne, E. (1992). A conceptual foundation for software re-engineering. *Proceedings Conference on Software Maintenance* (pp. 226-235). IEEE.

Clapis, J. (2015). *TSV427 Instructor's Guide.* The MITRE Corporation.

Cowley, J. (2014). *Job Analysis Results for Malicious Code Reverse Engineers: A Case Study.* CMU/SEI CERT.

Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering.* Indianapolis, IN: Wiley.

Endsley, M., & Models, M. (2000). Situation Models: An Avenue to the Modeling of Mental Models. *Proceedings of the 14th Triennial Congress of the International Ergonomics Association and the 44th Annual Meeting of the Human Factors and Ergonomics Society.*

Erdos, K., & Sneed, H. (1998). Partial Comprehension of Complex Programs (enough to perform maintenance). *Proceedings of the 6th International workshop on Program Comprehension* , (pp. 98-105).

Ericsson, K. A., & Charness, N. (1994, August). Expert Performance: Its Structure and Acquisition. *American Psychologist*, pp. 725-747.

Gladwell, M. (2008). *Outliers.* Little, Brown and Company.

Gobet, F. (2005). Chunking models of expertise: implications for education. *Applied Cognitive Psychology*, 183-204.

Goldstein, G., & Hersen, M. (2000). *Handbook of Psychological Assessment.* Elsevier.

Heelan, S. (2011, May/June). Vulnerability Detection Systems: Think Cyborg, Not Robot. *IEEE Security & Privacy*, pp. 74-77.

Miller, M., & DeWitt, S. (2014). *Reverse Engineering Survey 2014: Conclusions, Progress, and Recommendations DRAFT.* Bedford, MA: The MITRE Corporation.

Miller, M., Rogers, W., & Dewitt, S. (2014). *Reverse Engineering and Software Analyst Survey Results Version 1.2.* Bedford, MA: The MITRE Corporation.

Posluszny, F. (2015). *Foundations of Software RE Wiki.* MITRE.

Pozluszny, F. (2015). *MITRE Intro to Reverse Engineering Course.* MITRE.

Rajlich, V. (2009). Intensions are a key to program comprehension. *2009 IEEE 17th International Conference on Program Comprehension* (pp. 1-9). IEEE.

Soloway, E., & Ehrich, K. (1984). Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering* (pp. 595-609). IEEE.

Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., . . . Saxena, P. (2008). BitBlaze: A New Approach to Computer Security via Binary Analysis. *ICISS, LNCS* (pp. 1-25). Heidelberg: Springer-Verlag.

Stevens Institute of Technology. (2009). *Graduate Software Engineering 2009: Curriculum Guidelines for Graduate Degree Programs in Software Engineering.* Integrated Software and Systems Engineering Curriculum Project.

Storey, M.-A. (2005). Theories, Methods and Tools in Program Comprehension: Past, Present and Future. *13th International Workshop on Program Comprehension (IWPC '05)* (pp. 181-191). IEEE.

The Joint Task Force on Computing Curricula. (2004). *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.* IEEE Computer Society Association for Computing Machinery.

The Joint Task Force on Computing Curricula. (2013). *Computer Science Curricula 2013.* Association for Computing Machinery (ACM) IEEE Computer Society.

Tilley, S. (1998). *A Reverse-Engineering Environment Framework.* Pittsburgh, PA: Carnegie Mellon Software Engineering Institute.

Tucker, R. (2003). A Developmental Study of Cognitive Problems in Learning to Program. *Proceedings of Joint Conference EASE & PPIG*, (pp. 325-332).

Zayour, I., & Lethbridge, T. (2000). A cognitive and user centric based approach for reverse engineering tool design. *CASCON '00 Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative Research.* IMB Press.

Appendix A Abbreviations and Acronyms

ACM	Association of Computing Machinery
API	Application Program Interface
CADP	Cryptanalysis Development Program
CASL	Center for the Advanced Study of Language
CPU	Central Processing Unit
IEEE	Institute of Electrical and Electronics Engineers
MAEC	Malware Attribute Enumeration and Characterization
OJT	On the Job
OS	Operating System
RE	Reverse Engineering
SME	Subject Matter Expert
Y2K	Year 2000