Adding Uncertainty to Hypertext Models of Software Systems

Submitted to the International Workshop on Incorporating Hypertext Functionality into Software Systems

Hadar Ziv
Debra J. Richardson
Department of Information and Computer Science
University of California, Irvine
Irvine, California 92697-3425
{ziv,djr}@*ics.uci.edu*

Conceptually, hypertext offers substantial benefits for visualization and navigation of large software systems, as follows: Hypertext systems allow users to create, evolve, and examine networks of interlinked textual and multimedia information [HM94]. Similarly, software development may be viewed as creation (i.e., synthesis), evolution, and examination (i.e., analysis) of networks of interrelated artifacts. Consequently, software systems effectively comprise large hypertexts and therefore may be visualized and navigated as such [SZ92]. Also, as mentioned in [ZO95], software development readily meets Shneiderman's three "golden rules" of hypertext [Shn89], as follows:

1. A large body of information is organized into numerous fragments;
2. The fragments relate to each other; and
3. The user needs only a small fraction at any time.

Due to their complexity, software systems are extremely difficult to understand and, subsequently, to create, evolve, and maintain. Hypertext may be instrumental in addressing the challenging problem of software understanding, since:

- It offers a coherent conceptual metaphor of nodes and links that is applicable and relevant at all stages of the software lifecycle.
- Software systems are typically developed in distinct phases, by different teams of developers, and are delivered as collections of artifacts, including, among others, requirements specifications, user manuals, design documents, source code and executable releases, and test cases and test results. Often, these artifact collections are at best loosely connected within themselves and to each other. Artifact relations are often poorly understood and rarely captured explicitly and accurately. Hypertext offers the opportunity, perhaps for the first time, to view and navigate both inter- as well as intra-artifact relationships in consistent fashion.
- Most hypertext systems are armed with additional visualization and navigation tools, such as maps and overview diagrams, history and backtracking capabilities, and even fish-eye and cluster views. When applied to software development, such tools are expected to further enhance its understanding.

Despite the promising prospects above, successful application of hypertext to real-life software may still require significant efforts, particularly for legacy systems not originally designed for hypertext. In such cases, one must weigh the potential benefits of hypertext against the effort required by its provision. It is also important to consider whether hypertext capabilities, once available, will indeed be used effectively by intended users. We therefore contend that care and caution must be exercised in applying hypertext to software engineering, for the following reasons:

- Though most software systems induce a hypertext-like structure of artifacts and relations, many are not developed in a way that allows hypertext to be integrated seamlessly and unobtrusively. This is especially evident in legacy systems, where many software artifacts, not to mention relationships, are either undocumented, poorly archived, or simply gone.

- Hypertext cannot magically eliminate software complexities. Rather, complexity is generally considered an inherent and essential property of software development, for which no "silver bullet" exists [Bro87, ZR97]. Though hypertext offers a conceptual metaphor accompanied by useful tools, its benefits, as well as unforeseen perils, are yet to be evaluated in practice. We are especially concerned regarding the hypertext navigation problem known as "lost in hyperspace." According to Nielsen, users experience confusion and disorientation even when navigating relatively small hypertexts [Nie95]. It should be expected, therefore, that users will be easily confused and frequently lost in large and complex software spaces.

- In its general form, hypertext technology does not take into account the unique characteristics of software nor the special viewing and navigation needs of its users. Most software engineers, however, believe that software development has several unique characteristics that must be considered if hypertext is to be applied successfully. Furthermore, additional insight into software engineering processes and products may in fact be used to tailor and improve hypertext capabilities for software development. Several recent articles discuss unique considerations for providing hypertext for large scale software systems (cf. [Bie96]). Here, we consider the evaluation criteria of [Ore96]:

  - *Heterogeneous artifacts.* The hypertext should be able to capture different types of artifacts and their complex relationships, and allow for their dynamic incorporation.
  - *Link types and automatic link generation.* Links among software artifacts should be modeled explicitly; different link types and their semantics should be supported; and, when appropriate, provisions for automatic creation of links should be made.
  - *Multiple levels of anchor granularity.* The hypertext should support anchors at all levels of granularity, from entire documents to single source-code statements.
  - *Heterogeneous environment.* The hypertext must accommodate existing software development tools and integrate seamlessly and unobtrusively into existing environments.
  - *Scalability.* The hypertext must be scalable to large software systems and support viewing, editing and navigation by many users.

The promise and perils of hypertext are therefore summarized as follows:

- Hypertext, both as a coherent conceptual metaphor as well as a set of practically useful tools, holds great promise for improved software visualization, navigation and understanding.

- In practice, however, one should weigh and consider both the practical implications of providing hypertext as well as its benefits and efficacy for actual users.

- We are particularly concerned about user disorientation in large software spaces as well as the need to address various unique characteristics of the software endeavor.

To address the latter concerns, we are currently developing an approach for improving navigation in large software spaces by means of utilizing additional insight and information regarding the ongoing development process. Specifically, we consider it beneficial to model, explicitly and accurately, software engineering uncertainties that would otherwise be tacit, implied but not captured, or simply ignored. This approach is part of an overarching strategy for modeling software uncertainties, based on the following reasoning:

- Software development is fraught with uncertainties, which are intimately tied to its complexities. Software uncertainties may lead to software risks, maintenance nightmares, symptoms of the software crisis, and to software–related accidents.

- We proposed the Maxim of Uncertainty in Software Engineering (MUSE) [ZR97], stating that *uncertainty is inherent and inevitable in software development processes and products.*

- We also proposed, as a corollary to MUSE, that *software uncertainties should be modeled and managed explicitly.*

We also suggested that a particular technique for uncertainty modeling, called Bayesian belief networks (originally defined by Pearl [Pea88]), be used for software uncertainties. Perceived benefits of applying the Bayesian approach include:

- Bayesian networks, like hypertext and software systems, also form a complex web of nodes and links. This supports a coherent conceptual correlation among the architecture of hypertext, software systems and Bayesian networks.

- Bayesian networks allow for uncertainties to be initially captured and later revised dynamically. This allows for uncertainty values to be updated and maintained as software development progresses.

- Bayesian networks support multiple sources of evidence, i.e., they accommodate multiple factors that may affect uncertainty values. This is deemed beneficial since multiple sources of uncertainty, both internal as well as external, typically exist for most software systems. Sources of software uncertainties include, among others, problem domain uncertainties, solution domain uncertainties, and human-involvement uncertainties (cf. [ZR97]).

We are studying the feasibility of the Bayesian approach by applying it to a real system currently under development at Beckman Instruments in Fullerton, California. This system, called CEquencer, controls and communicates with hardware devices and is used by biologists, chemists, and other scientists to separate laboratory specimens into molecular constituents to help determine their DNA sequences.

So far we have created a network model for requirements, design, code and test artifacts for Beckman's CEquencer software. Bayesian "belief" values have been established for a subset of CEquencer artifacts by eliciting them from Beckman developers and domain experts. A sample network of requirements, test, and code artifacts is provided below. Our preliminary experiences with practical application of the Bayesian approach seem to confirm that:

- The underlying network model of hypertext, software systems, and Bayesian networks offers a convenient metaphor for visualization, navigation, and understanding. It is straightforward to model software as a web of artifacts as long as sufficient information regarding artifact relationships is retained.

- Visualizing software as hypertext offered a convenient navigational metaphor. Specifically, early depictions of artifact webs were developed as Java applets and put on the World Wide Web (see [Ziv97]); these applets were then used by Beckman developers to view and comment on the depictions' accuracy and relevance.
- The CEquencer project, like most other software projects, is fraught with uncertainties, thereby confirming the MUSE maxim for at least one real-life system. Specifically, CEquencer exhibits many, often subtle, problem domain uncertainties. These include uncertainties stemming from laws of physics and chemistry in the software's operational environment (for instance, "depending on the choice of enzyme, temperature settings for chemical reactions should be in the $\{t1..t2\}$ range"), as well as from vaguely defined boundaries between software versus hardware components.
- Furthermore, Beckman developers and domain experts were comfortable with the notion of "software uncertainties" and were readily able to describe their beliefs regarding CEquencer software artifacts using probability values or "degrees of confidence."

Figure 1 shows an example network of software artifacts together with associated uncertainty values. We are now poised to complete the elicitation of belief values from CEquencer developers, followed by evaluation of the efficacy of the Bayesian approach for visualization, navigation, and understanding of the entire CEquencer system.

References

[Bie96] Michael Bieber. Second workshop on incorporating hypertext functionality into software systems, 1996. WWW Document http://eies.njit.edu/ 333/htf/htf2_cfp.html.

[Bro87] Frederick P. Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, April 1987.

[HM94] Frank Halasz and Schwartz Mayer. The dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, February 1994.

[Nie95] Jakob Nielsen. *Multimedia and Hypertext : the Internet and beyond.* AP Professional, Boston, MA, 1995.

[Ore96] Peyman Oreizy. Hypertext for software engineering environments, December 1996. UC Irvine (ICS 280) Term paper.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[Shn89] Ben Shneiderman. Reflections on authoring, editing, and managing hypertext. In E. Barret, editor, *The Society of Text*, pages 115–131. MIT Press, Cambridge, MA, 1989.

[SZ92] Dani Steinberg and Hadar Ziv. Software Visualization and Yosemite National Park. In *Proceedings of the Twenty-Fifth Annual Hawaii International Conference on System Sciences*, January 1992.

[Ziv97] Hadar Ziv. Java applets for Beckman CEquencer software, March 1997. WWW Document http://www.ics.uci.edu/ ziv/java/bayesian_example.html.

[ZO95] Hadar Ziv and Leon J. Osterweil. Research issues in the intersection of hypertext and software development environments. In Richard N. Taylor and Joëlle Coutaz, editors, *Software Engineering and Human-Computer Interaction*, volume 896 of *Lecture Notes in Computer Science*, pages 268–279. Springer-Verlag, Berlin Heidelberg, 1995.

[ZR97] Hadar Ziv and Debra J. Richardson. Bayesian-network confirmation of software testing uncertainties. Technical report, University of California, Irvine, January 1997. submitted to ESEC/FSE'97.

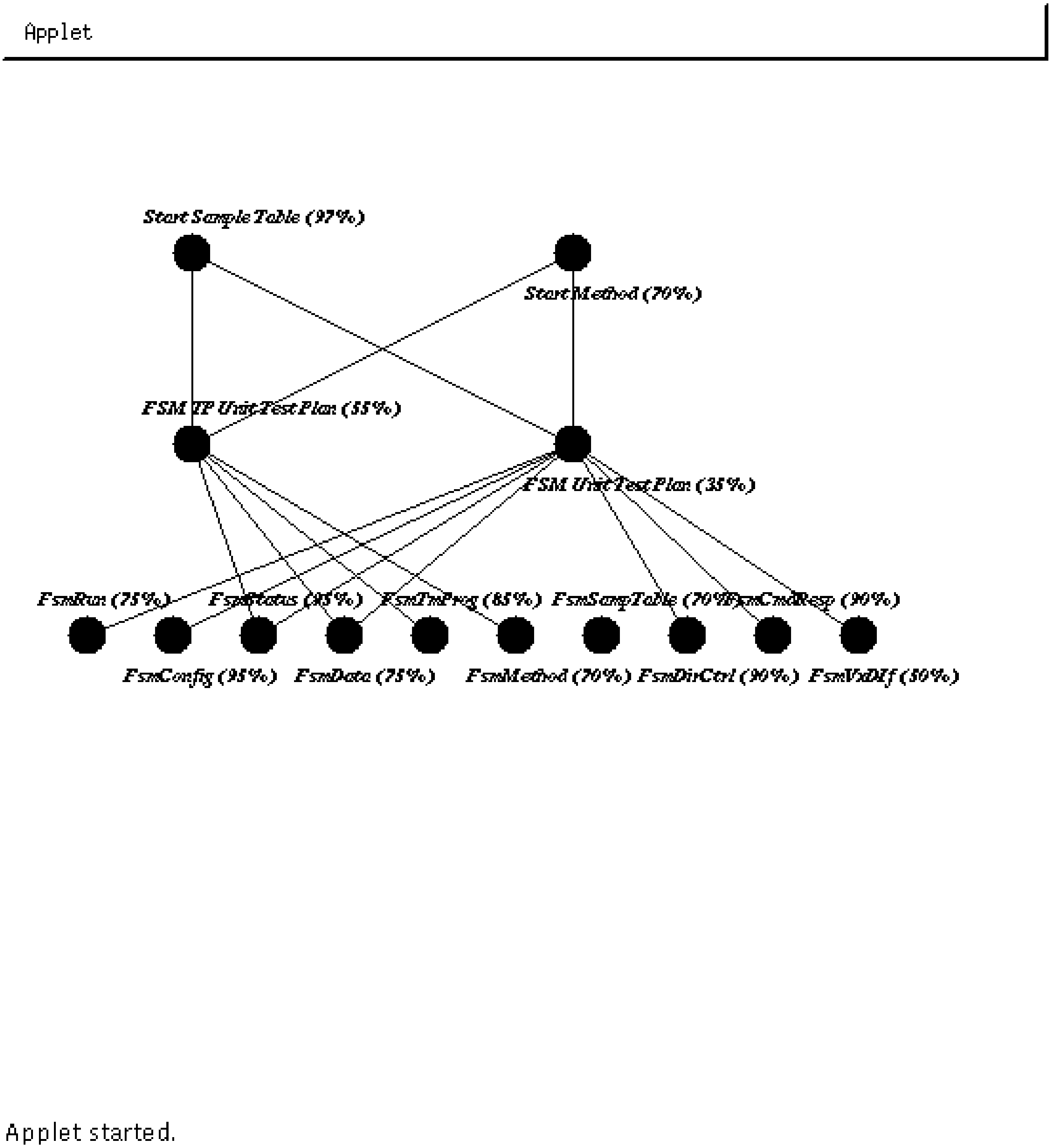

Figure 1: A Java Applet Depicting CEquencer Software Uncertainties