

Narrative Structures for Process-Level Reasoning in Software Engineering

Md. Hasan Tareque

Dept. of Mathematics and CS
University of Lethbridge
mdhasan.tareque@uleth.ca

John Anvik

Dept. of Mathematics and CS
University of Lethbridge
john.anvik@uleth.ca

Abstract

Software engineering students often struggle to reason about how development activities interact across the software lifecycle. Requirements, design, implementation, and testing are frequently learned as isolated topics, leading to fragmented process-level mental models. This work-in-progress paper explores whether narrative structures can function as cognitive scaffolds for software engineering reasoning. We propose an instructor-guided framework in which large language models generate short instructional narratives aligned with software engineering concepts. The framework maps narrative structures such as the Hero's Journey, the Fichtean Curve, and the Three-Act Structure onto software engineering reasoning tasks involving lifecycle progression, architectural evolution, and testing processes. The paper presents a framework for using narrative-based representations to support process-level reasoning and outlines future classroom evaluation.

1. Introduction

Software engineering students are expected to reason about how requirements, design, implementation, testing, deployment, and maintenance activities interact across the software lifecycle (Pressman, 2005; Sommerville, 2016). However, these topics are often taught separately, making it difficult for students to connect decisions made in one stage with consequences that appear later in development.

Many degree programmes address this fragmentation through semester-long or two-semester team software projects. These projects provide valuable opportunities for students to experience multiple development activities within a shared context. Nevertheless, participation in an integrated project does not necessarily ensure that students explicitly recognize the causal relationships among those activities. Project scope and duration may also limit students' exposure to later lifecycle activities such as deployment, long-term maintenance, and architectural evolution.

This challenge is especially important in areas such as DevOps, maintenance, testing, and architecture, where students must reason about trade-offs, dependencies, and cascading consequences. Learners may understand individual concepts, or participate in projects involving several development activities, while still struggling to explain how decisions made during one activity influence outcomes elsewhere in the software process.

This work-in-progress paper examines whether narrative structures can support process-level reasoning in software engineering education. Rather than using stories primarily for engagement, the proposed framework examines whether narrative forms can help learners organize software engineering processes through structured narrative progression.

The framework maps narrative structures such as the Hero's Journey, the Fichtean Curve, and the Three-Act Structure onto software engineering reasoning tasks. Large language models are used to generate short instructional narratives under instructor guidance, allowing scalable creation of process-oriented scenarios while maintaining pedagogical control.

The paper presents a conceptual framework rather than an empirical evaluation. It introduces a framework for using narrative structures to support process-level reasoning in software engineering education and outlines a future exploratory classroom study suitable for investigation within the PPIG community.

2. Related Work and Motivation

Research on program comprehension shows that learners may struggle to integrate software structure, behaviour, intent, and dependencies into coherent mental models (Pennington, 1987; Letovsky, 1986; Johnson-Laird, 1983). Experts tend to organize knowledge around higher-level concepts, whereas novices rely more heavily on surface features (Adelson, 1981; Chi, Feltovich, & Glaser, 1981; Petre, 2013; Storey, 2005). This distinction matters in software engineering, where reasoning must extend across requirements, design, implementation, testing, and maintenance. When these topics are presented separately, the consequences of decisions made in one activity may remain difficult to trace in another (Sommerville, 2016; Sharp, Dittrich, & de Souza, 2009). The growing emphasis on process thinking, systems reasoning, and AI-supported instruction further increases the need for representations that connect activities across the lifecycle (Marlowe, Ku, Laracy, Kirova, & Herbert, 2026).

Narrative cognition offers a possible foundation for such representations. People use narratives to organize events, actions, and consequences over time (Bruner, 1991; Schank, 1995), while situation-model research explains how readers construct causal and temporal coherence from connected events (Graesser & Bower, 1980; van Dijk & Kintsch, 1983; Trabasso & van den Broek, 1985). Research on external cognition likewise shows that representational form can influence reasoning by exposing relevant relationships, distributing cognitive work, and directing attention (Larkin & Simon, 1987; Zhang & Norman, 1994; Scaife & Rogers, 1996). Narrative may therefore provide a complementary way to represent how software processes develop through interacting decisions and constraints.

Evidence from technical practice supports this possibility. Orr’s ethnographic study of copier technicians found that diagnostic stories preserved contextualized experience, supported collaborative interpretation of symptoms and tests, and contributed to a shared community model (Orr, 1986). Those stories emerged naturally in professional work, unlike the instructor-designed narratives proposed here, but they demonstrate that narrative can participate in technical reasoning rather than merely describe it.

Story-based approaches have also been used in computing education through contextualized programming, storytelling environments, and scenario-based learning (Kelleher & Pausch, 2007; Rankin & Barnes, 2009; Pierson, Kinnunen, & Simon, 2022). Erwig’s *Once Upon an Algorithm*, for example, uses familiar situations to explain algorithms, data structures, abstraction, recursion, searching, and unsolvability (Erwig, 2017). Its focus is primarily on individual computing concepts, whereas the present work considers relationships among software engineering activities across time.

Related studies have examined narrative interpretation of code behaviour and software processes (Dey, Powell, & Eijkhout, 2025), as well as interactive stories intended to provide contextualized and inclusive learning experiences (Ghailani, Malaise, & Signer, 2025). Much of this literature emphasizes explanation, engagement, accessibility, or motivation. This paper instead investigates narrative structures as representations for process-level reasoning and mental-model construction.

Developments in Artificial Intelligence in Education and LLM-based support now make varied instructional scenarios easier to produce (Kasneci et al., 2023; Holmes & Porayska-Pomsta, 2022). Here, however, the LLM serves only as an instructor-guided generation mechanism. The central contribution is the use of narrative structure to organize reasoning about relationships across the software lifecycle.

3. Narrative Structures as Process Representations

The proposed framework reorganizes software engineering topics into narrative scenarios that emphasize progression, dependency relationships, and consequences across development activities.

The framework is instructor-guided. Instructors select the software engineering topic, determine the learning objective, and decide how narrative structures should map onto development activities. Large language models are then used to generate short instructional scenarios from these mappings. The goal is not automated instruction, but scalable generation of contextualized examples under pedagogical oversight. Figure 1 summarizes the proposed workflow.

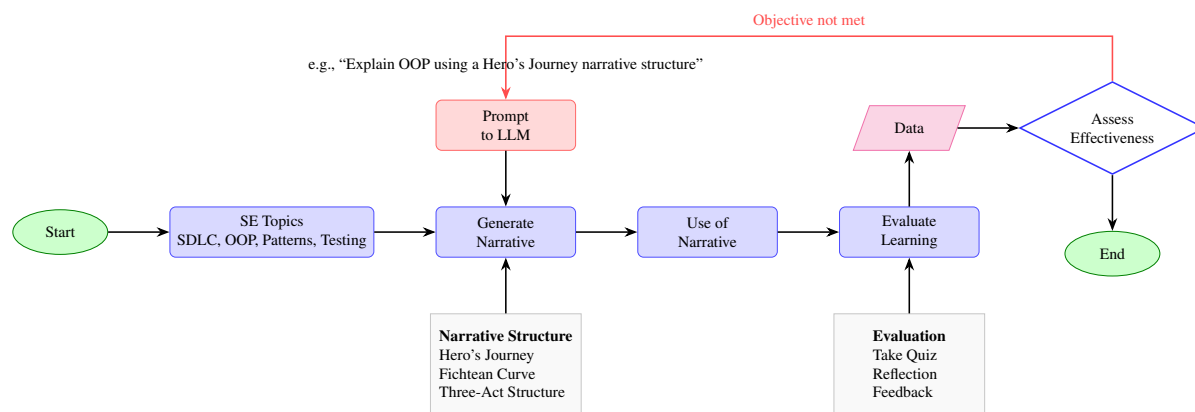


Figure 1 – Instructor-guided narrative generation workflow.

The current version of the framework focuses on three narrative structures: the Hero’s Journey, the Fichtean Curve, and the Three-Act Structure. These structures were selected because they provide contrasting ways of organizing change over time: transformation across an extended journey, escalation through accumulating pressures, and resolution of a bounded problem. The proposed mappings are based on three considerations. First, there should be *structural correspondence*: transitions between narrative phases should correspond to meaningful changes in the software process rather than merely provide decorative story elements. Second, the structure should foreground the *causal relationship* relevant to the learning objective, such as progression, accumulation, or diagnosis. Third, the mapping should have a distinct *pedagogical purpose* by drawing attention to relationships that may be less explicit in a conventional lifecycle diagram or topic-by-topic presentation.

The Hero’s Journey provides a temporal scaffold for lifecycle progression. Requirements and stakeholder needs establish the problem; design, implementation, and iteration transform the system; and testing, deployment, and maintenance reveal the effects of earlier decisions. The analogy does not portray development as a linear heroic quest, but highlights progression and iterative returns across lifecycle activities.

The Fichtean Curve foregrounds the pressures that drive design evolution. Changing requirements, technical debt, and interacting constraints increase tension until the existing design requires restructuring or refactoring. Figure 2 illustrates this progression through a late authentication requirement: local workarounds increase technical pressure until redesign becomes necessary, after which testing determines whether stability has been restored.

The Three-Act Structure frames testing and debugging as movement from expectation to disruption and verified resolution. The setup defines expected behaviour and test conditions, while confrontation introduces failures and unexpected interactions. Resolution combines correction, validation, and regression testing, showing how evidence revises the learner’s explanation of system behaviour.

These mappings are neither exhaustive nor exclusive. A testing scenario could, for example, be represented through a Fichtean Curve when failures accumulate across several releases, while architectural evolution could be represented through a Three-Act Structure when the learning objective concerns a single redesign episode. The appropriate mapping therefore depends on the relationship learners are expected to reason about, rather than on a fixed association between a narrative form and a software engineering topic. At this work-in-progress stage, the mappings should be understood as theoretically motivated design hypotheses whose usefulness and limitations require empirical investigation.

The mappings shown in Figure 3 organize software engineering concepts as evolving process relationships rather than as isolated instructional topics. Together, they distinguish three reasoning demands: tracing progression across phases, explaining how constraints accumulate, and diagnosing how a disruption

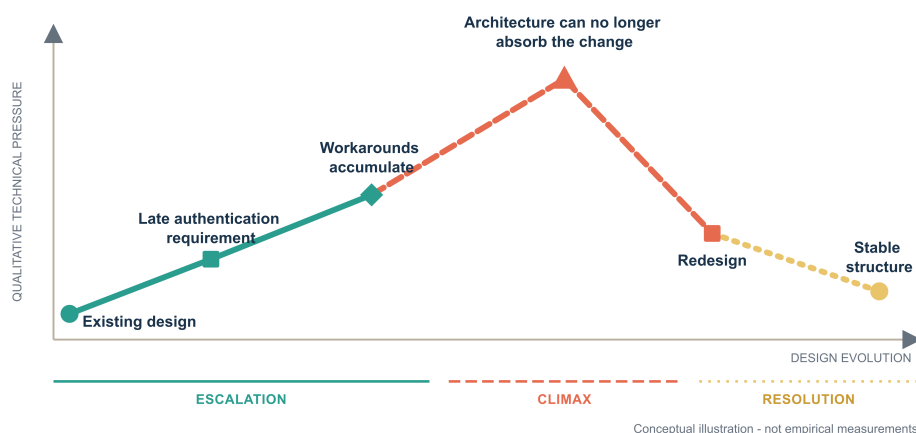


Figure 2 – Conceptual application of the Fichtean Curve to design evolution. A late authentication requirement and accumulating workarounds increase technical pressure until redesign becomes necessary. The vertical axis is qualitative rather than empirically measured.

tion is resolved. From a cognitive perspective, these representations may support temporal organization by representing development activities as connected event sequences, causal reasoning by linking decisions with downstream effects, and process simulation by helping learners mentally trace how software processes evolve over time.

These mappings are intended to support reasoning about process relationships rather than factual recall. A testing failure, for example, may be framed as the consequence of earlier requirements, design, or implementation decisions. Similarly, refactoring may be represented not simply as a technique, but as a response to accumulated design tension. In this way, narrative scenarios can complement UML diagrams, workflows, and lifecycle models by foregrounding temporal progression, causal propagation, and downstream effects.

A simple example illustrates the representational shift. A conventional instructional description might present a late requirements change as a sequence of separate development activities. A workflow diagram can show the order of activities, but it does not by itself show why the later testing problem occurred.

Requirements change → architecture modification → implementation updates → regression testing failures

Within the proposed framework, the same situation can be represented narratively:

A development team introduces authentication requirements late in the implementation phase. Existing architectural assumptions no longer support the new requirements, requiring significant redesign. The resulting implementation changes introduce unexpected interactions that later appear as regression test failures during deployment preparation.

The underlying phenomenon represented in both versions is the same software development process. What changes is the representational form. The conventional version lists activities and outcomes, whereas the narrative version presents them as a story involving actors with a goal, a disruption to an existing situation, decisions made in response, and consequences that emerge over time. In this distinction, *process* refers to the actual relationships among software engineering activities, while *story* refers to the instructional representation used to make those relationships explicit. The story should therefore preserve the relevant technical mechanisms rather than replace them with merely dramatic events.

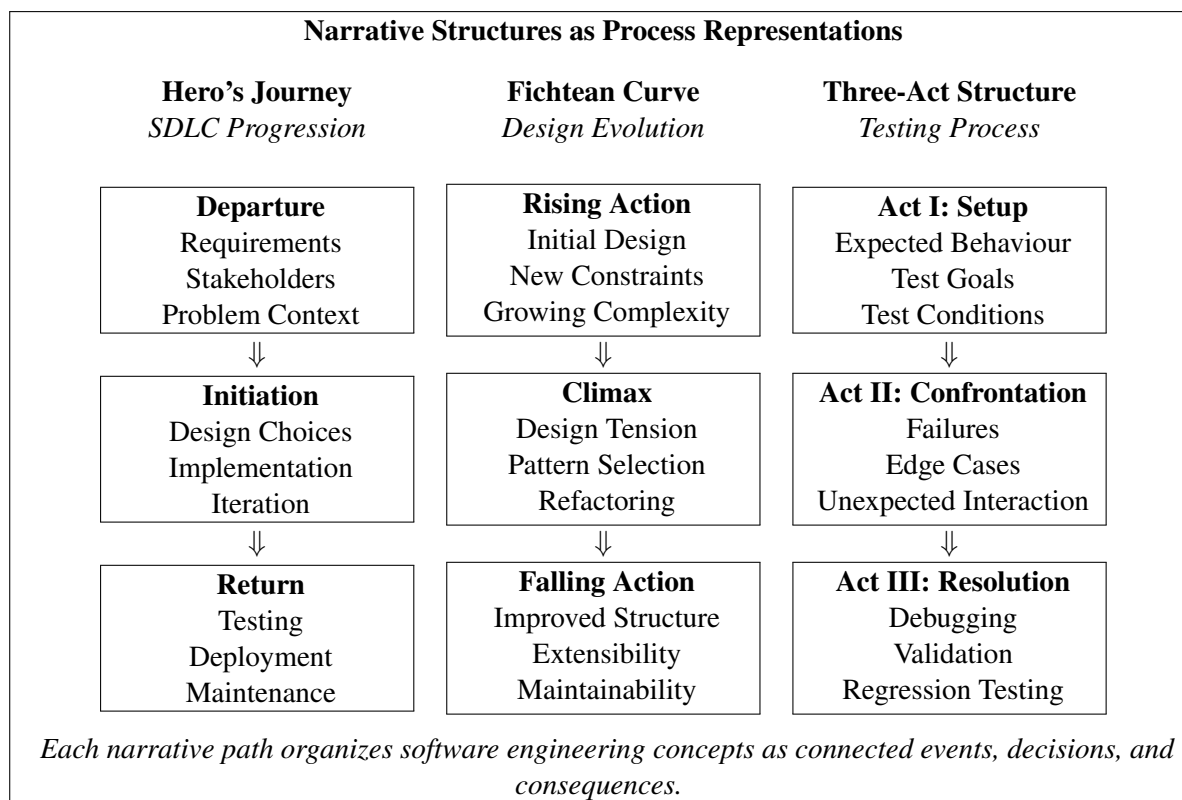


Figure 3 – Narrative structures as representations of software engineering reasoning paths.

In the example, the late authentication requirement creates pressure on architectural assumptions that were originally designed for a simpler interaction model. The redesign may introduce new abstractions, changes to database access patterns, and modifications to testing infrastructure. These changes can then produce integration issues and regression failures that were not visible when the original implementation decisions were made. The narrative form foregrounds the causal chain by connecting the team's goal and decisions to the later outcomes; the process itself remains the sequence of software engineering activities and dependencies being represented.

The narrative transformation reorganizes software engineering concepts into connected process explanations intended to represent how software processes evolve through interacting decisions, constraints, and outcomes. Narrative structures act as process-oriented representations that may help learners reason about how software engineering activities influence one another over time.

This is consistent with situation model theory, where learners build an internal representation of how events, actions, and consequences are connected (van Dijk & Kintsch, 1983; Trabasso & van den Broek, 1985). This framing aligns with interests in program comprehension, mental models, and representational support for reasoning.

Instructional narratives are generated through an instructor-guided process aligned with software engineering concepts and learning objectives. Instructors select target topics, choose narrative structures appropriate for the intended reasoning tasks, and develop prompts that align narrative phases with software engineering activities.

LLMs are then used to generate short instructional narratives that students engage with alongside lectures, scenario-based exercises, and classroom discussions intended to support process-level reasoning.

4. Discussion and Future Work

This framework treats narrative structures as instructional representations for reasoning about software engineering processes. This framing shifts attention away from narratives as engagement devices and toward narratives as representational forms that may shape how learners organize, interpret, and reason about software engineering processes.

One implication of this work is that software engineering education may benefit from additional representational forms beyond diagrams and workflow models. Conventional representations such as life-cycle diagrams, workflow models, and UML diagrams are valuable because they make structural and procedural relationships visible. However, they do not always foreground the temporal, causal, and consequential nature of software engineering decisions. Narrative structures instead organize development activities as sequences of events in which earlier decisions create later consequences.

The framework also raises several questions for future work in software engineering education and the psychology of programming. If narrative structures support process-level reasoning, then future work should examine which narrative features matter most: temporal progression, causal linking, conflict, agency, consequence, or resolution. It should also examine when narrative representations are less effective than formal representations, particularly for topics requiring precision, notation, or formal constraint reasoning.

Future work will empirically investigate whether narrative-based representations influence how learners reason about software engineering processes and relationships across development stages. A planned exploratory classroom study will compare narrative-based instructional scenarios with conventional process-oriented representations using scenario-based reasoning tasks. The study will examine how learners explain dependencies, consequences, and progression across software engineering activities, with particular attention to whether narrative representations support more coherent process-level explanations.

5. Conclusion

This work-in-progress paper explored whether narrative structures can function as cognitive scaffolds for process-level reasoning in software engineering education. The proposed framework treats narrative forms as instructional representations that organize software engineering concepts through progression, conflict, and consequence. By mapping structures such as the Hero's Journey, the Fichtean Curve, and the Three-Act Structure onto software engineering reasoning tasks, the framework aims to support more coherent mental models of how development activities interact across the software lifecycle.

The paper contributes a conceptual framework for integrating narrative-based representations with instructor-guided LLM-generated instructional scenarios. Rather than replacing lifecycle models, workflows, or UML diagrams, the framework explores how narrative structures may complement them by making process relationships and downstream effects more visible to learners.

Future work will investigate whether narrative-based instructional representations influence learners' ability to reason about progression, dependencies, and consequences across software engineering activities. More broadly, this work contributes to ongoing PPIG discussions surrounding mental models, program comprehension, and the role of external representations in supporting software engineering reasoning.

6. Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery Grants program.

Generative AI tools were used to assist with language refinement and editing during the preparation of this paper.

7. References

- Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition*, 9(4), 422–433.
- Bruner, J. (1991). The narrative construction of reality. *Critical Inquiry*, 18(1), 1–21.
- Chi, M. T. H., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5(2), 121–152.
- Dey, S. C., Powell, J. H., & Eijkhout, V. (2025). Coding through storytelling: Narrative reasoning and software engineering education. *Journal of Computational Science Education*, 16(2), 5–9. doi: 10.22369/issn.2153-4136/16/2/2
- Erwig, M. (2017). *Once Upon an Algorithm: How stories explain computing*. The MIT Press.
- Ghailani, I. G., Malaise, Y., & Signer, B. (2025). JsStories: Improving social inclusion in computer science education through interactive stories. *arXiv preprint arXiv:2504.04006*. Retrieved from <https://arxiv.org/abs/2504.04006>
- Graesser, A. C., & Bower, G. H. (1980). Story understanding as problem solving. *Poetics*, 9(1–3), 223–250.
- Holmes, W., & Porayska-Pomsta, K. (2022). The ethics of AI in education: Towards a community-wide framework. *International Journal of Artificial Intelligence in Education*, 32(3), 504–526.
- Johnson-Laird, P. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard University Press. Retrieved from <https://books.google.ca/books?id=FS3zSKAFLGMC>
- Kasneci, E., et al. (2023). ChatGPT for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103, 102274.
- Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM*, 50(7), 58–64.
- Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1), 65–100.
- Letovsky, S. (1986). Cognitive processes in program comprehension. In *Proceedings of the first workshop on empirical studies of programmers* (pp. 58–79).
- Marlowe, T. J., Ku, C. S., Laracy, J. R., Kirova, V. D., & Herbert, K. G. (2026). A systemic view of a software engineering education curriculum: Requirements and guidelines in the era of generative AI. *Journal of Integrated Design and Process Science*. doi: 10.1177/10920617251405471
- Orr, J. E. (1986). Narratives at work: Storytelling as cooperative diagnostic activity. In *Proceedings of the 1986 acm conference on computer-supported cooperative work* (pp. 62–72). ACM. doi: 10.1145/637069.637077
- Pennington, N. (1987). Comprehension strategies in programming. *Empirical Studies of Programmers: Second Workshop*, 100–113.
- Petre, M. (2013). Expertise and collaboration in software engineering. *2013 35th International Conference on Software Engineering (ICSE)*, 1397–1400.
- Pierson, E., Kinnunen, P., & Simon, B. (2022). Storytelling as a pedagogical tool in computer science education. *ACM Transactions on Computing Education*, 22(4), 1–25. doi: 10.1145/3494145
- Pressman, R. S. (2005). *Software engineering: A practitioner's approach* (6th ed.). McGraw-Hill.
- Rankin, Y., & Barnes, T. (2009). Contextualized gaming for learning. In *Proceedings of the 4th international conference on foundations of digital games* (pp. 244–251).
- Scaife, M., & Rogers, Y. (1996). External cognition: How do graphical representations work? *International Journal of Human-Computer Studies*, 45(2), 185–213.
- Schank, R. C. (1995). *Tell me a story: Narrative and intelligence*. Northwestern University Press.
- Sharp, H., Dittrich, Y., & de Souza, C. (2009). *The role of ethnographic studies in empirical software engineering*. Springer.
- Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.
- Storey, M.-A. (2005). Theories, methods and tools in program comprehension: Past, present and future. In *13th international workshop on program comprehension (IWPC'05)* (pp. 181–191).

doi: 10.1109/WPC.2005.38

- Trabasso, T., & van den Broek, P. (1985). Causal thinking and the representation of narrative events. *Journal of Memory and Language*, 24(5), 612–630.
- van Dijk, T. A., & Kintsch, W. (1983). *Strategies of discourse comprehension*. Academic Press.
- Zhang, J., & Norman, D. A. (1994). Representations in distributed cognitive tasks. *Cognitive Science*, 18(1), 87–122.