

# Software Cinema—Video-based Requirements Engineering for Agile Design

Oliver Creighton<sup>†</sup>, Martin Ott<sup>‡</sup>, Bernd Bruegge<sup>‡</sup>

<sup>†</sup>*Siemens AG, CT SE 1*

<sup>‡</sup>*Technische Universität München  
[creight, ott, bruegge]@in.tum.de*

November 15, 2005

## Abstract

Designing and implementing mobile distributed interactive applications represents a challenge to the communication between end-users and system specialists. Heavy-weight techniques of model-driven design do not offer representations exhibiting enough agility for revolutionary changes, expressiveness of visions, and directness of communication. This may lead to misunderstandings about the actual requirements of potential end-users.

The video-based requirements engineering technique “Software Cinema” addresses these challenges by employing digital video technology to develop films that are used in the software development process. Using a Software Cinema tool kit, filmmakers can assemble a collection of scenes that can be traversed in differing sequential orders to examine various possible scenarios of an envisioned system. These films represent differing, possibly contradictory, points-of-view on the part of various stakeholders, in particular the potential end-users. Additionally, they represent alternative, branching event sequences based on differing use cases. Films can represent a reality that is not yet realisable—a visionary scenario of what is desired.

Individual film segments display the real-time sequence of events that are pertinent to the system. These real-time segments may be paused or viewed in slow-motion as dictated by communication and modeling needs. The Software Cinema technique provides two capabilities: First, film segments can be embedded in documents, commentaries, annotations, and software artifacts. Second, software artifacts can be embedded directly within the film. This allows issues and design rationale discussions to be attached directly to elements of the target environment of the application. Films are used for requirements engineering to enhance communication between all stakeholders and to enrich agile, model-based software development. Analysts and developers can acquire a deeper understanding of the application domain and actual end-user requirements. This bridges the gaps between requirements, design, and manufacturing.

The Software Cinema technique has three phases: 1) Preproduction, which leads to a visionary scenario. 2) End-user session, in which feedback is incorporated. 3) Postproduction, where analysis leads to enriched formal requirements. We distinguish three roles: Customer/End-user, who commissions a project and

who is a domain specialist; Analyst, who is in charge of requirements elicitation and analysis; Video producer, who shoots, edits, and modifies scenario films with standard video editing, compositing, and animation tools.

The technique is based on a semiology-based mapping of metamodels for software and film. To support it, a tool kit for digital video editing and annotation is used. The tool kit is based on an ontological model that is extensible and customizable as dictated by concrete requirements needs.

Events and their temporal relationships can be explicitly modeled to create a clear specification of timing and sequencing requirements. Participating actors and objects can be annotated and their spatial relationships can be specified on a fine-grained level of detail, if necessary.

## 1 INTRODUCTION

Innovation begins and ends with people: Before a visionary system can be described, stakeholders in the entire product lifecycle need to be identified. As market success is notoriously hard to predict, to better understand the people it involves seems like a particularly promising idea. [Nor03] A focus on customers and end-users and their early involvement in a development project is a cornerstone of agile methods. [CH01] Many small design decisions influence the final look and feel of products, but unless a representative group of end-users is involved at all times, the result can be suboptimal. However, this might only be realized when a product is released to the public.

To aid inventors in assessing the scope of the context in which new devices might be used in, use case driven models of the application domain have been suggested. [JCJO93] In contrast to technically possible devices, the clear focus on actors aids in guiding inventors to valid models of the future application domain. [Jac95] Ideally, the very same people whose domain is in question validate these models. The validation of models, however, requires knowledge and expertise with notations and semantics. Moreover, various competing notations exist today, so this expertise is hard to acquire and maintain.

The Unified Modeling Language (UML) [Obj03] is gaining a lot of industry-support, but consequently allows many variations and customizations. Its intended goal of unifying notational conventions, however, is thus endangered for the dubious benefit of code generation or other edge cases of model-driven design. By following the agendas of tool vendors and by enabling highly specialized applications of UML right in its specification, its complexity is almost prohibitive of becoming a ‘lingua franca’ and may rather lead to another confusion of tongues in software modeling.

Scalability requirements of software processes, involving ever more people in various roles, have aggravated the situation. This is in large parts due to the need for decomposition of large and complex systems to manageable chunks that can be manufactured by single developers. This model hierarchy, however, has not always brought the desired effect that all stakeholders can quickly and easily grasp the scope of the entire system. Even in state-of-the-art notations, models are still expressed in languages designed for solution domain experts. In particular, for systems that consist of many diverse and dynamically changing components, maybe even used in unforeseen environments, we still lack a system representation that will tell the *story* and paint the *big picture*. [NE00]

## 2 APPROACH

The Software Cinema technique uses motion pictures as a semi-formal representation of software models. This addresses two issues: First, providing a model of reality that all stakeholders can understand equally well. Second, giving all involved developers a rich base of reference for what the complete system is intended to achieve. The technique describes how to create and extend a digital video-based *description* of the application domain in order to provide an *analysis* of actual end-user requirements.

Film is used as a very natural model of reality. [Mon00] The informality of film enables modeling scenarios that still contain inconsistencies, which is an important feature when gathering requirements from multiple viewpoints. Additionally, film is particularly well suited to express system behavior over time.

The technique is based on digital video, as this allows to create and use sophisticated tools for manipulation and annotation of the footage. By making the film ‘clickable,’ i.e. allowing a viewer to directly refer to objects seen on screen, the envisioned reality that is depicted can be discussed and annotated on a fine-grained level of detail.

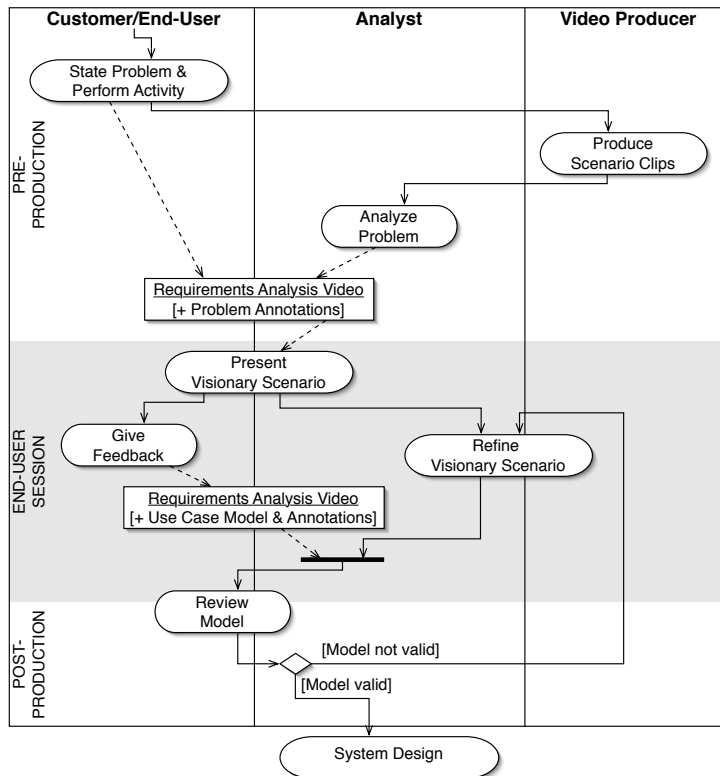
These are important steps towards a valid model of the application domain. Rich discussions about trade-off situations provide the rationale that designers and developers need when designing and manufacturing a product. The relatively high effort that is required for a video-based requirements analysis can easily be justified by comparing its added value to the development process: Video-based models can be reused for entire families of product lines and provide entry-points for new and highly innovative products. The effort, in turn, is put into perspective in the first place by comparing how much more effort is needed to fix a mistake of the requirements specification in later phases.

## 3 The Software Cinema Technique and Tool Kit

The Software Cinema technique has three phases: Preproduction, end-user session, and postproduction. The preproduction phase contains all activities that lead to a visionary scenario that can be discussed with end-users and other stakeholders. We distinguish three roles in the Software Cinema technique. The customer/end-user commissions a project and also provides the domain knowledge. The analyst, a member of the developer organization, is in charge of the requirements analysis. A new software engineering role of video producer is in charge of shooting the video clips.

After shooting the clips, they are annotated in multiple ways by the analyst, focusing on developer’s needs. A basic annotation is to identify pixel regions in the clip and assign identifiers to them, so the the movie becomes ‘clickable’ and objects that are seen can be selected directly on screen.

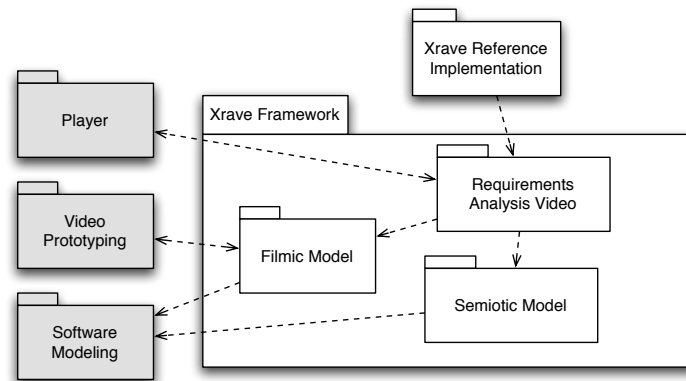
After enough film material has been produced to make a discussion with the end-users feasible, the analyst shows the initial visionary scenario, based in a real-life situation that end-users are familiar with. During the discussion, the analyst can directly select and annotate the semantic units.



When end-users are satisfied with how the Requirements Analysis Video shows the system to be like, the end-user session phase can be concluded. The analyst can then create the use case model that is added to the Requirements Analysis Video and also validated by the end-user.

In the postproduction phase, the analyst has many video clips that show various scenarios of the system. Furthermore, several alternative takes, exceptional, and forbidden behavior is also available in the Requirements Analysis Video. When end-users agree to the use case model in the Requirements Analysis Video, it is handed over to manufacturing together with the system specification. Otherwise a new iteration is initiated.

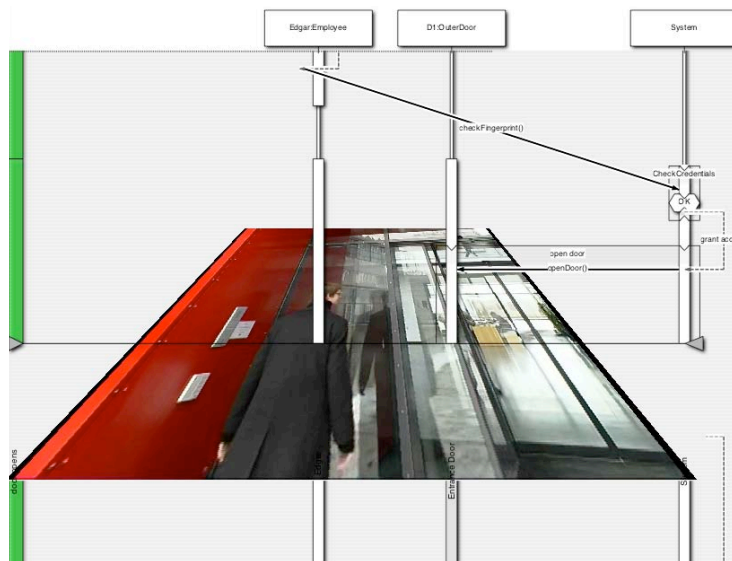
To enable the Software Cinema technique, we defined and developed the Software Cinema tool kit. It supports innovation driven by end-users as discussed by von Hippel and Katz [vHK02]. To realize our vision—using videos as models for software development processes—the tool kit requires appropriate tools for video prototyping and software modeling.



The Software Cinema tool kit consists of a number of commercial off-the-shelf components that must be treated as blackbox components: These are the player, video prototyping, and software modeling components. At its heart, there is the Xrave framework, providing abstractions for combining film and Software Cinema models. A reference implementation of a Software Cinema tool for editing Requirements Analysis Videos, called Xrave, uses these abstractions and indirectly the commercial off-the-shelf components. Requirements Analysis Videos consist of UML diagrams (or extensions thereof) and video footage. These two elements can be linked semantically to facilitate the validation of software models by end-users, who only need to understand the video.

The link between software and film metamodels is achieved by applying a semiology-based mapping. Semiotics—the study of signs—defines a sign as the basic unit of signification composed of the *signifier* and the *signified*. In the context of Software Cinema, signifiers denote audio-visual video elements. Signifiers can be composited to express relationships between them. Each signifier encodes a certain signified meaning, its signified. Such signified meanings are, at first hand, abstract concepts. Software Cinema specifies a translation of signified meanings to software model primitives and so maps software and film metamodels.

An experimental presentation mode is the ‘three-dimensional’ viewpoint of the sequence editor shown in the following figure. When watching a video, the spectator looks at the video from roughly a 45° angle above. In other words, the video is projected onto a virtual screen laying in front of the spectator like a ‘light table.’ The sequence chart is drawn as usual—upright—on the real screen. There is only one video frame visible at a time, but the action and spatial relationships between objects can be followed, nevertheless. The current movie time in the diagram is where the video layer intersects the diagram. During playback, the object boxes and their lifelines follow the positions of the signifiers in the video and the events are moving from bottom to top. This way, past events can be seen above and future events below the video layer.



## 4 DISCUSSION

How may Software Cinema aid in bridging the gap between design and manufacturing? When we contemplate state-of-the-practice specifications that are handed over from design to manufacturing, we can identify four main challenges: consistency, missing context, missing rationale, and general information structure.

**1) Consistency:** Specifications of considerable size usually suffer from so-called ‘Thesaurus’-problems. Even elaborate project glossaries may not entirely control the effect of synonyms (different words for the same concept) and homonyms (identical words for different concepts). This may be reduced when one eliminates the need for written or spoken words: The video of a concept simply shows the concept, no matter the language or the expressions used to describe it. Therefore, common (identical) entry points are provided for all stakeholders. Software Cinema allows adding forward traces from scenario films to specifications. Developers, who get access to the films together with the specifications, are intuitively able to follow these traces. The common entry points allow easy access to the design of the proposed system and thereby aid in avoiding overspecification, which is a main source of inconsistencies.

**2) Context:** Due to the pragmatic limitation of scope of specifications for individual sub-systems, each development team receives only small pieces of information. Finding the right amount of context for a specification of a sub-system is hard. This is often due to the fact that the persons authoring the specification may either lack this information themselves or that condensing without losing the information is too much effort in time-constrained design. Particularly in model-based software specifications, information about the context and the environment in which the proposed system shall be deployed is fully abstracted away. Scenario films present the proposed system embedded into its environment and clarify the interactions with this environment. Thus, film gives the developers a broader view of the system and the environment it is inter-

acting with. Ambiguity or inappropriate abstraction levels can be resolved by developers, because the rich context on film helps them to interpret the specification.

**3) Rationale:** Discussions with end-users rarely revolve around solution domain concepts. Rather, end-users are inclined to point out facts about changes in their application domain that are based on their mode of thinking. There may be an impedance mismatch when attaching the arguments for and against certain trade-offs to a model of the solution. Using film language, complex relationships can be made explicit in a model of the application domain. This allows to capture the reasons for design choices in a better-suited decomposition of the problem. By showing the effects of design alternatives all the way up to their consequences on the scenario films, end-users are empowered to meaningfully discuss alternatives.

**4) Structure:** As all stakeholders in a Software Cinema-based project share identical high-level material—the Requirements Analysis Video—they can, on the one hand, infer common knowledge and, on the other hand, drill down into the specification as needed to fulfill their particular tasks. By purposefully using a ‘fuzzy’ model like film to tie together various entry-points into the complete set of specifications, Software Cinema allows different roles to view the set from perspectives that are appropriate individually yet synchronized across the board. Negotiations of this high-level representation may reveal important (mis-)conceptions individual stakeholders have. At the same time, the various viewpoints may actually be reconciled on this level, as the technique does not force an ‘absolute truth’ on all stakeholders.

Software Cinema is, in our opinion, a useful technique that contributes to solving the major challenges for requirements engineering as discussed by Nuseibeh and Easterbrook [NE00]: modeling and analysing the application domain, bridging the gap between contextual and formal approaches, providing a rich model for non-functional requirements, understanding the impact of software architectural choices, reuse of requirements models, and multidisciplinary requirements engineering. We have conducted an exploratory empirical evaluation which confirmed our expectations of the technique and provided guidance on how to improve both, the technique and the tool kit.

## 5 CONCLUSION

The focus of our research is to validate a new theory that combines film and software models to aid requirements engineering in agile design processes. The theory provides a foundation, basic design, and inspiration for the defined Software Cinema tool kit and its custom-built Xrave tool for the creation of ‘Requirements Analysis Videos.’ The tool kit definition also includes a guideline for how to use commercial off-the-shelf components in a Software Cinema project. In essence, the theory regards video as a model that is closest to end-users and offers guidance as to how this model maps to computer-based implementations of innovative products. We assume that requirements analysis generally works by abstraction. To facilitate better communication between all stakeholders from end-user to manufacturing, the Software Cinema technique allows rich, broadband knowledge transfer on appropriate levels of abstraction. This should ultimately lead to higher-quality products that have been built by incorporating several viewpoints and globally optimized trade-off decisions.

## References

- [BD03] Bernd Bruegge and Allen H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns and Java*. Prentice-Hall, New Jersey, 2nd edition, 2003.
- [CH01] Alistair Cockburn and Jim Highsmith. Agile software development: The people factor. *IEEE Computer*, 34(11):131–133, November 2001. Available from: <http://doi.ieeecomputersociety.org/10.1109/2.963450> [cited June 2005].
- [Cre05] Oliver Creighton. *Software Cinema — Employing Digital Video in Requirements Engineering*. PhD thesis, Institut für Informatik der Technischen Universität München, June 2005. (Preliminary Version).
- [HM03] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, August 2003.
- [IKR00] Giulio Iacucci, Kari Kuutti, and Mervi Ranta. On the move with a magic thing: Role playing in concept design of mobile services and devices. In *Proceedings of the Conference on Designing Interactive Systems (DIS 2000)*, pages 193–202. ACM Press, 2000.
- [Jac95] Michael Jackson. *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. Addison-Wesley Publishing Company, 1995.
- [JCJO93] Ivar Jacobson, M. Christerson, P. Jonsson, and G. Oevergaard. *Object-oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing Company, New York, 4th edition, 1993.
- [Mon00] James Monaco. *How to Read a Film: The World of Movies, Media, and Multimedia: Art, Technology, Language, History, Theory*. Oxford University Press, New York, 3rd book & DVD edition, March 2000.
- [MRJ00] Wendy E. Mackay, Anne V. Ratzer, and Paul Janeczek. Video artifacts for design: Bridging the gap between abstraction and detail. In *Proceedings of the Conference on Designing Interactive Systems (DIS 2000)*, pages 72–82. ACM Press, 2000.
- [NE00] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In *ICSE — Future of SE Track*, pages 35–46, 2000.
- [Nor03] Donald A. Norman. *Emotional Design: Why We Love (Or Hate) Everyday Things*. Basic Books, December 2003.
- [Obj03] Object Management Group. OMG unified modeling language specification [online]. March 2003 [cited July 2004]. Available from: <http://www.omg.org/technology/documents/formal/uml.htm>.
- [vHK02] Eric von Hippel and Ralph Katz. Shifting innovation to users via toolkits. *Management Science*, 48(7):821–833, July 2002.