In Proceedings of ASWEC'97, Sydney, Sept 28-Oct 2, 1997, IEEE CS Press.

Utilising Past Event Histories in a Process-Centred Software Engineering Environment

John C. Grundy[†], Warwick B. Mugridge[††] and John G. Hosking[††]

Department of Computer Science[†]
University of Waikato
Private Bag 3105, Hamilton
New Zealand
jgrundy@cs.waikato.ac.nz

Department of Computer Science[††]
University of Auckland
Private Bag, Auckland
New Zealand
{rick, john}@cs.auckland.ac.nz

Abstract

When working on complex software systems, it is often difficult for multiple software developers to coordinate their work, and for developers to coordinate their multiple tool and software process usage. Process-centred Software Engineering Environments attempt to help developers manage the complexities of such coordination by codifying steps in a software process, and codifying the "work context" a developer utilises (i.e. the artefacts, tools and collaborators the developer requires during their work). Unfortunately most process-centred environments do not adequately support work coordination, ease-of-use and improvement of process models. We describe our work utilising histories of past events within a process-centred environment to give developers extra leverage when using process models to guide collaborative software development. We describe techniques for work history determination, improved visualisation support for work coordination, and automatic process enactment and process improvement. Our approach to realising these facilities within a process-centred software engineering environment is described, and our experiences using our event history-based techniques during software development is reviewed.

1. Introduction

When working on large-scale software development projects, developers are faced with difficulties coordinating their work with others, coordinating their use of multiple development tools, and adhering to a (sometimes loosely defined) "process" for carrying out their work [8, 20]. Developers (either implicitly or explicitly) have a "work context", which is made up of the software artefacts they are building, tools they are using to build these artefacts, communication tools and some notion of the part of the development process they are working on. Work contexts become large and complex, even for small projects. Difficulties in coordinating work, managing complex work contexts, and adhering to defined processes leads to problems during development, and ultimately to poor quality of software [3, 27].

Work contexts may be fixed for specific phases of a well-defined software process, such as detailed system design and coding, and such codified work contexts help in understanding others' work, particularly the inter-relationships between work being carried out in parallel [12]. At other times a developer's work context may be volatile, with a varying set of artefacts, tools and/or collaborators over time. Coordinating the work of several developers without explicit work contexts and processes, and without suitable tool support, is very difficult. However, a requirement for explicit codification under such volatile conditions results in inflexible work practices and the use of inadequate process models [28].

A developer's "work context" is implicit in most software development tools. CASE tools and programming environments allow developers to view and modify a variety of diagrams and code, with those being edited or used to form the work context of the developer. While many support collaborative work via shared workspaces and/or version control and merging, most do not support higher-level work coordination schemes [21], limiting their effectiveness.

Process-Centred Environments (PCEs) allow developers to codify the processes used to develop software, and to codify their work contexts for each process stage. This use of codified work processes and contexts leads to better software development processes and thus to better software quality [3, 4, 5, 28]. Most PCEs require developers to explicitly specify which artefacts, tools and/or collaborators are utilised for each stage in a software process model [1, 2, 28], and do not support the determination of work contexts from past work. This results in limitations which reduces their degree of work coordination support, and thus their ability to contribute to improved software production and production processes:

- As many stages of development involve volatile work contexts, developers may not wish or may not even be able to formally specify exactly what artefacts, tools,

collaborators and communication mechanisms they want to use for all process stages [27, 19, 23].
- Visualisation of collaborators' work contexts is usually limited, with often no visualisation of the current work contexts of process stages, and no visualisation of past work contexts of a stage. This leads to less effective work coordination.
- Most PCEs utilise a notion of a "current" enacted process model stage, which can be overly-restrictive, as a developer may have several process model stages enacted at once. This necessitates regular, manual switching between them.
- Process improvement in most PCEs does not adequately utilise the history of work done on a project to feedback into the process improvement process, limiting the degree of process improvement a developer can achieve with the PCE.

We describe our work in utilising the history of work contexts for a process stage, and the history of work by a developer, to provide better support for work coordination, automatic process stage enactment, and process improvement within a PCE and its related, integrated tools. Our techniques help to reduce the management load in multiple-developer projects by automatically tracking work context changes, automatically keeping developers aware of each others current, past and likely future work contexts, and in limited automatic enactment of appropriate process stages based on prior event histories.

2. Related Research

Most CASE tools, such as Software thru Pictures [30], and Integrated Software Development Environments, such as Dora [24], FIELD [25], and SPE [10], do not support the idea of codifying or determining work contexts for software development. This means that while they support the processes of doing work on a software project, they do not assist developers in the complex tasks of work coordination using multiple people, tools and processes.

Most workflow systems provide some manner of codifying the context of work for each process in the workflow. This is either just the roles involved, as in Regatta [28] and TeamFLOW [29], or more explicit artefact, tool and role definitions as in Action Workflow [22]. Some, such as TeamFLOW, provide process stage enactment histories, but do not record artefact, tool and communication events. Once again, this limits the degree of work coordination the tools can provide for developers, and work history is not used in improving the processes.

PCEs such as SPADE [3], ProcessWEAVER [8], EPOS [7], E^3 [1], ADELE-TEMPO [4], and Oz [5] require explicit codification of aspects of a process stages work context, such as the artefacts and tools it uses. These systems provide limited visualisation of enacted process models, with the enacted stages being highlighted. However, no support for work context highlighting is provided nor usage of work histories during process improvement. This again limits the degree to which they can effectively support large project collaborative work.

Groupware systems, such as GroupKit [26] and Rendezvous [18], provide various mechanisms for "synchronous" group awareness ie. keeping collaborators aware of artefacts in current use, but not things previously used or likely to be used ("asynchronous" group awareness). Some CSCW systems, such as wOrlds [6] and Orbit [19], allow work contexts to be implicitly defined by the artefacts, tools, actions and collaborators involved, rather than explicitly codified. However, most groupware systems still suffer from lack of flexibility in defining a of determining process stage work contexts, and still tend to force people to be in only one context at a time.

3. The Serendipity PCE

In the following sections, we describe facilities we have added to the Serendipity PCE to better support work coordination and process improvement for software development using prior event histories. These techniques help alleviate some problems with current PCEs, workflow systems and groupware applications.

Serendipity is a process modelling, enactment and work planning environment, which also supports event handling, group communication, and group awareness facilities [16, 12]. Fore example, the left and bottom windows shown in Figure 1 are Serendipity views that model part of the ISPW6 software process example [20]. Stages (rounded rectangles) describe steps in the process of modifying a software system. Enactment event flows link stages, where the label is the finishing state of the stage the flow is from. Serendipity also supports artefact, tool and role modelling. Usage connections show how stages, artefacts, tools and roles are related. Serendipity also provides graphical filters (rectangular icons) and actions (ovals), which process arbitrary enactment and work artefact modification events [16, 11, 12]. Work coordination in the ISPW6 model is defined by the top-right window of Figure 1. The filters detect when testing starts or completes, and actions inform the project manager of this. Artefact updates made OOA/D diagrams are stored in a changes list owned by the project manager.

We have integrated Serendipity with SPE (Snart Programming Environment) [10], an integrated software development environment, to produce SPE–Serendipity [12, 13]. Serendipity provides the process modelling and enactment capabilities used to guide or enforce work in SPE. SPE provides object-oriented analysis, design, implementation, debugging and documentation views of software development [10]. Figure 2 shows a screen dump from SPE–Serendipity. The bold, shaded stage aff.2.2:Review Design Changes is the "current enacted stage" for developer "judy". Each developer may have several process model stages enacted at one time, but only one "current" enacted stage (the one they are doing work on at the present moment).

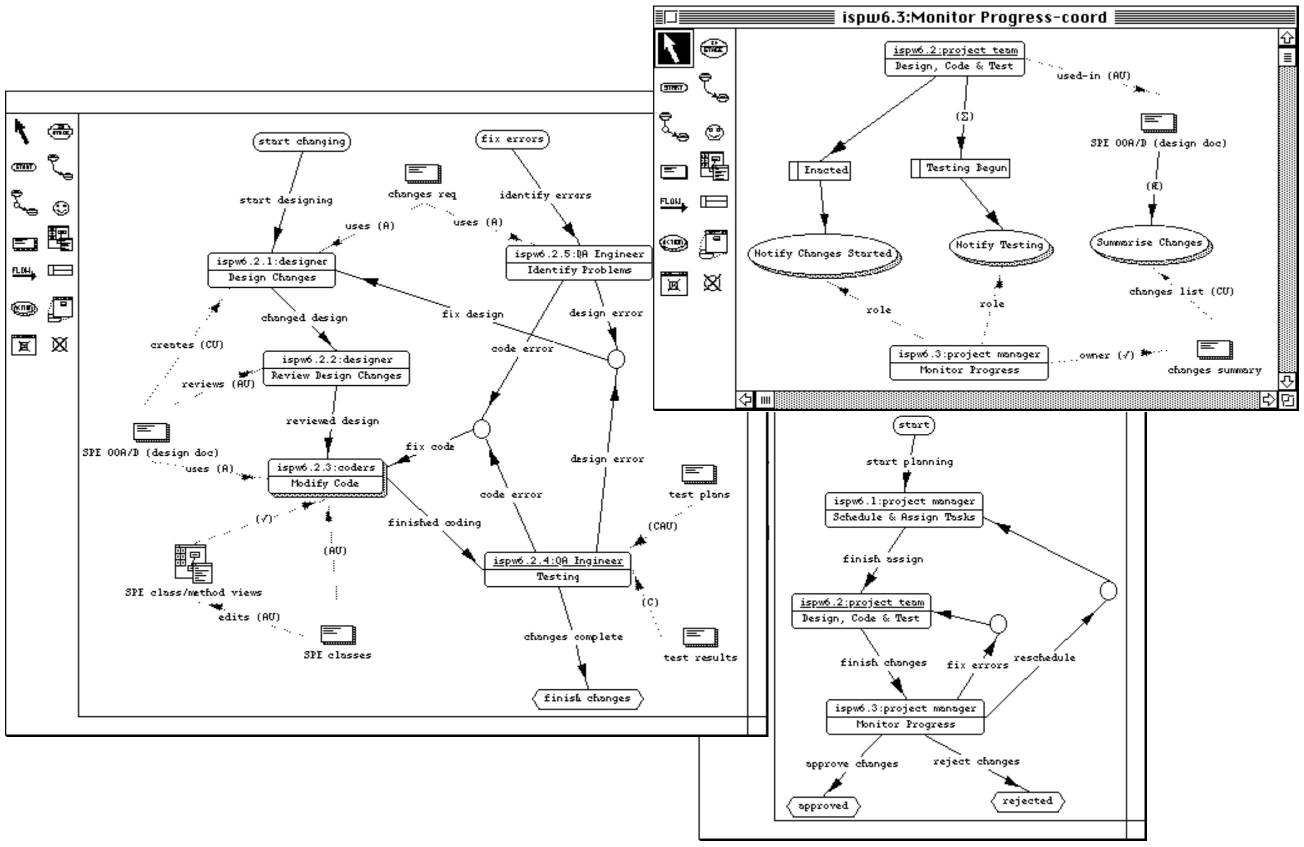


Figure 1. Part of the ISPW6 software process example modelled in Serendipity.

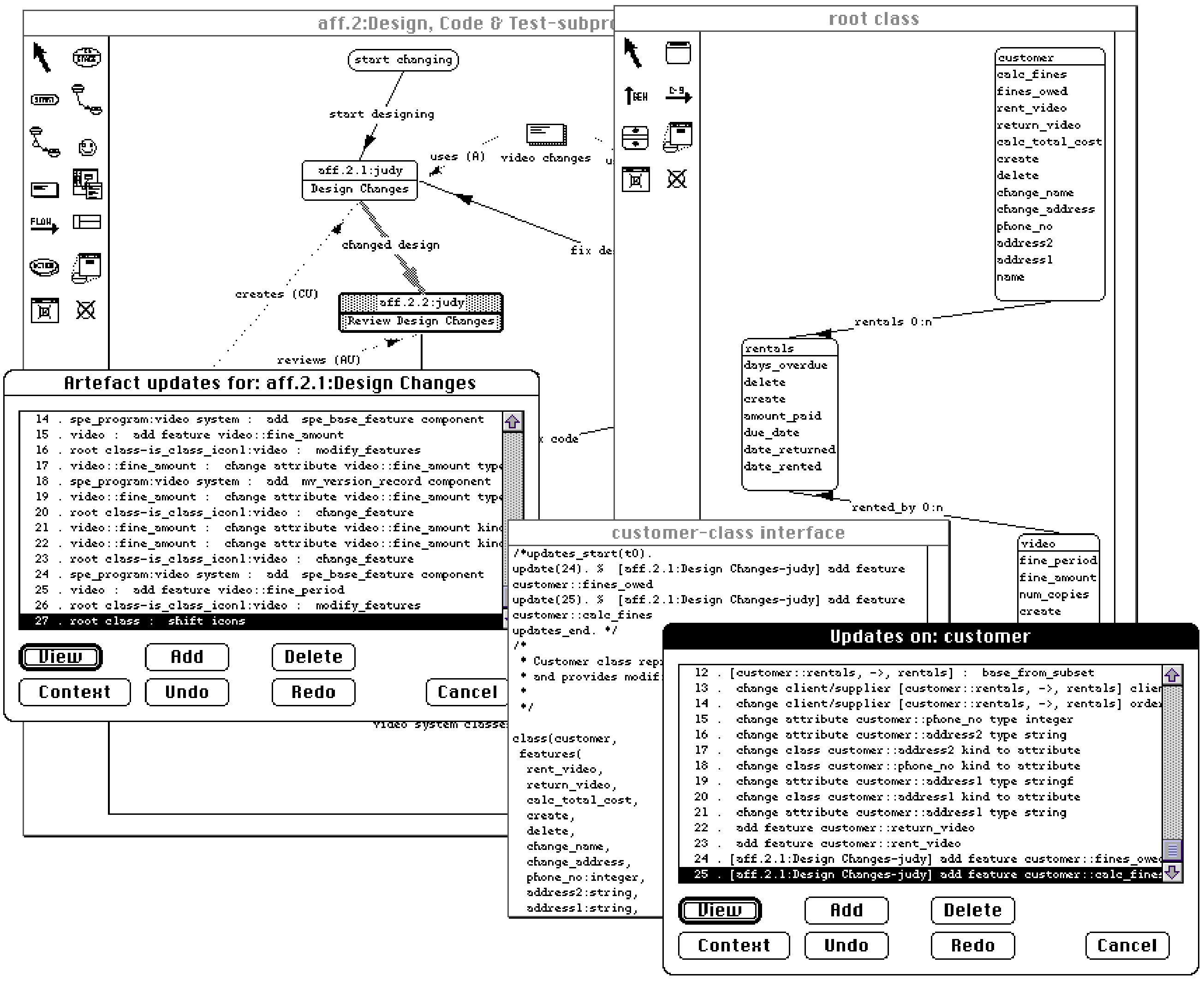
"Change descriptions", which document all events occurring in SPE-Serendipity, are generated when enacting or finishing process model stages, when switching the current enacted stage, when making artefact modifications in SPE, when SPE tools generate events, and when developers communicate. Enactment events are stored by Serendipity for each stage, forming an enactment history. Change descriptions generated by SPE are augmented with information about the current enacted process model stage [12, 16], and are stored by Serendipity in artefact change histories for the current enacted stage, forming work histories for each software process stage. Figure 2 shows examples of artefact histories (left hand dialogue), augmented change descriptions being presented in SPE (middle textual view) and artefact modification histories with augmented change descriptions in SPE (right-hand dialogue).

4. Determining Work Contexts

Recorded change descriptions for stages and developers can be used to formalise the work context and process model a developer has been using (or may use in the future), and for developers to visualise the dynamic aspects of work context as they change over time. We have made extensions to SPE–Serendipity which support recording and collation of past events, so event histories can be used to provide improved group awareness and process enactment and improvement facilities.

Serendipity records change descriptions sent from SPE, which describe artefact updates and tool events. We can use these events to *determine* work contexts for process model stages and record work histories for particular developers. Figure 3 illustrates the process used to determine the artefacts, tools and collaborators someone is working with for some process stage. Whenever the developer accesses or modifies an artefact, or communicates with a collaborator, the tool they are using generates an event, and this is processed to record information in a work context database.

We have built reusable filter/actions which determine the work contexts for specified stages. A developer connects an appropriate filter/action icon to the process model stage icon for which they want work context information determined. These filter/actions record not only the artefacts/tools/people of interest, but also frequency of access, whether artefacts were created, updated, accessed or deleted, access/update times, etc.



```
14 . spe_program:video system :  add  spe_base_feature component
15 . video :  add feature video::fine_amount
16 . root class-is_class_icon1:video :  modify_features
17 . video::fine_amount :  change attribute video::fine_amount typ
18 . spe_program:video system :  add  mv_version_record component
19 . video::fine_amount :  change attribute video::fine_amount typ
20 . root class-is_class_icon1:video :  change_feature
21 . video::fine_amount :  change attribute video::fine_amount kin
22 . video::fine_amount :  change attribute video::fine_amount kin
23 . root class-is_class_icon1:video :  change_feature
24 . spe_program:video system :  add  spe_base_feature component
25 . video :  add feature video::fine_period
26 . root class-is_class_icon1:video :  modify_features
27 . root class :  shift icons
```

```
/*updates_start(t0).
update(24). %  [aff.2.1:Design Changes-judy] add feature
customer::fines_owed
update(25). %  [aff.2.1:Design Changes-judy] add feature
customer::calc_fines
updates_end. */
/*
 * Customer class rep
 * and provides modif
 *
 */

class(customer,
 features(
  rent_video,
  return_video,
  calc_total_cost,
  create,
  delete,
  change_name,
  change_address,
  phone_no:integer,
  address2:string,
  address1:string,
```

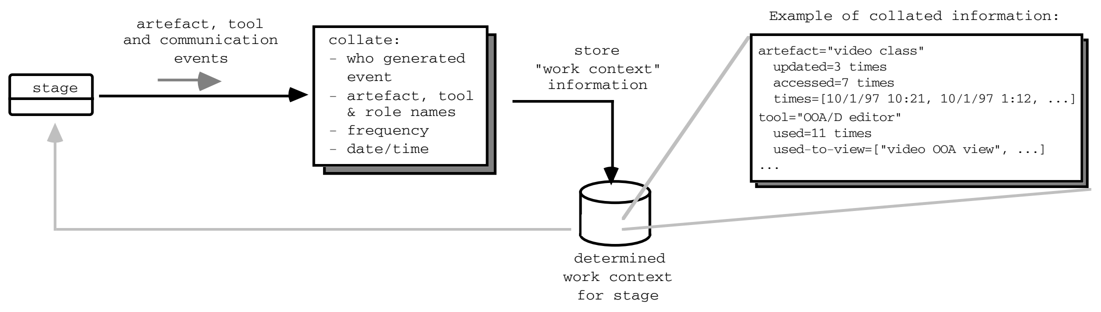
```
12 . [customer::rentals, ->, rentals] :  base_from_subset
13 .  change client/supplier [customer::rentals, ->, rentals] clie
14 .  change client/supplier [customer::rentals, ->, rentals] orde
15 .  change attribute customer::phone_no type integer
16 .  change attribute customer::address2 type string
17 .  change class customer::address2 kind to attribute
18 .  change class customer::phone_no kind to attribute
19 .  change attribute customer::address1 type stringf
20 .  change class customer::address1 kind to attribute
21 .  change attribute customer::address1 type string
22 .  add feature customer::return_video
23 .  add feature customer::rent_video
24 . [aff.2.1:Design Changes-judy] add feature customer::fines_owe
25 . [aff.2.1:Design Changes-judy] add feature customer::calc_fine
```

Figure 2. An example of work context capture and presentation in SPE–Serendipity.

We can also store all of the events generated by a particular developer, and record these in a database to form a work history for the developer, as shown in Figure 4. This information can then be presented to other software developers so they are made aware of the recent work of this developer. Such events can be filtered so only events pertaining to particular stages, tools or artefacts of interest are stored or presented.

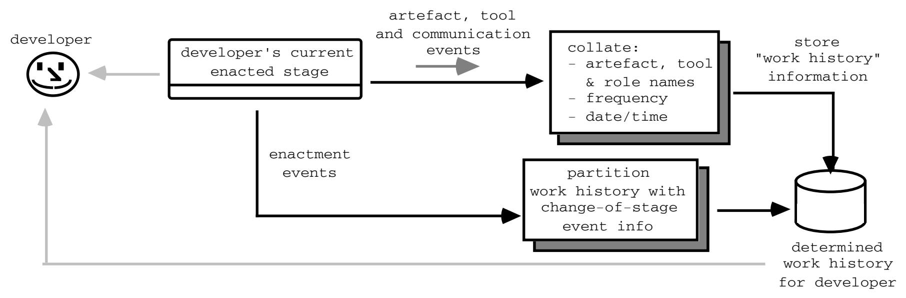
Storing the determined work contexts of process model stages and the work history of developers allows us to estimate the likely composition of future work contexts for both developer and process model stages. As shown in Figure 5, the current enacted stage, this stage's determined work context, and the recent work history for a developer are utilised to estimate the likely artefacts to be modified by the developer in the near future. Such information can be presented to collaborators so they can be made more aware not only of others' recent work history, but also their likely future actions.

Currently we apply simple analysis to the developer's work history and stage's determined work context to estimate the artefacts, tools and developers likely to be used by the developer in the near future. Items in the developer's work history are weighted based on whether they belong to the current enacted stage's determined work context, how recently the developer last used them, and if the developer has used particular items in sequence. The highest-weighted item names and weightings are recorded in a "likely work context" database for the developer.



stage
artefact, tool and communication events
collate:
- who generated event
- artefact, tool & role names
- frequency
- date/time
store "work context" information
Example of collated information:
artefact="video class"
updated=3 times
accessed=7 times
times=[10/1/97 10:21, 10/1/97 1:12, ...]
tool="OOA/D editor"
used=11 times
used-to-view=["video OOA view", ...]
...
determined work context for stage

Figure 3. Determination and collation of process model stage work contexts.



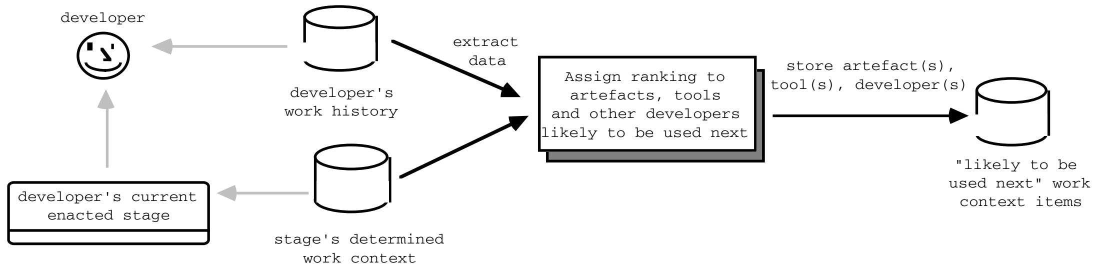
developer
developer's current enacted stage
artefact, tool and communication events
collate:
- artefact, tool & role names
- frequency
- date/time
store "work history" information
enactment events
partition work history with change-of-stage event info
determined work history for developer

Figure 4. Determination and collation of a developer's work history.

5. Visualising Work Contexts

To fully utilise the information accrued by the above work context and work history determination techniques, developers need mechanisms to access and visualise it. This enables developers to analyse their own and other developers' history of work and work contexts, to be made more aware of their collaborators' work, the reasons for it, any likely impact on their own work, and to determine if they need to coordinate their current work with them.

Figure 6 shows examples of Serendipity and SPE view icons which have been highlighted by our visualisation facilities. This screen dump is from designer judy's perspective, with her current enacted stage (aff.2.1) highlighted, as well as those of her collaborators john (aff.2.3) and rick (aff.2.5). Highlighting shows the artefacts currently in use (determined as the last things accessed/updated in their work histories) for judy and john. The shadowed artefact and tool icons in the left-hand (Serendipity) view are items in the determined work contexts for stages aff.2.1, aff.2.3 and aff.2.4. i.e. things that judy, john and rick have used when these stages were previously their current enacted stages. The shaded aff.2.4. stage icon indicates that this was the previous current enacted stage for rick (now no longer enacted), determined by examining rick's work history. The bold highlighted video class icon in the right-hand (SPE) view is being modified by john. The shadowed rentals class icon in the right-hand window of Figure 6 is an artefact predicted to be used by john again in the near future, based on events in his recent work history and the determined work context of stage aff.2.3. The highlighting used in these views allows judy to remain aware of items which are of recent, current and likely immediate future interest to her collaborators.



developer
developer's work history
extract data
Assign ranking to artefacts, tools and other developers likely to be used next
store artefact(s), tool(s), developer(s)
"likely to be used next" work context items
developer's current enacted stage
stage's determined work context

Figure 5. Estimating the future work contexts and work plans for a developer from their work history.

Visualisations can be used to summarise the work contexts of collaborators (determined from stage work contexts and from developer work histories). The left-hand side view in Figure 7 shows a simple visualisation using the determined work context for process stage aff2.1:Design Changes. This shows the actual artefacts, tools and collaborators that were used whenever developer judy had this stage enacted.

Developers request visualisations by connecting visualisation filter/action icons to artefacts representing the views in which they want icons highlighted. The visualisations can be tailored by the developer specifying highlighting mechanisms (eg. colour, shading); we have used only shading above to make this screen dump more readable when printed, but colouring is usually used.

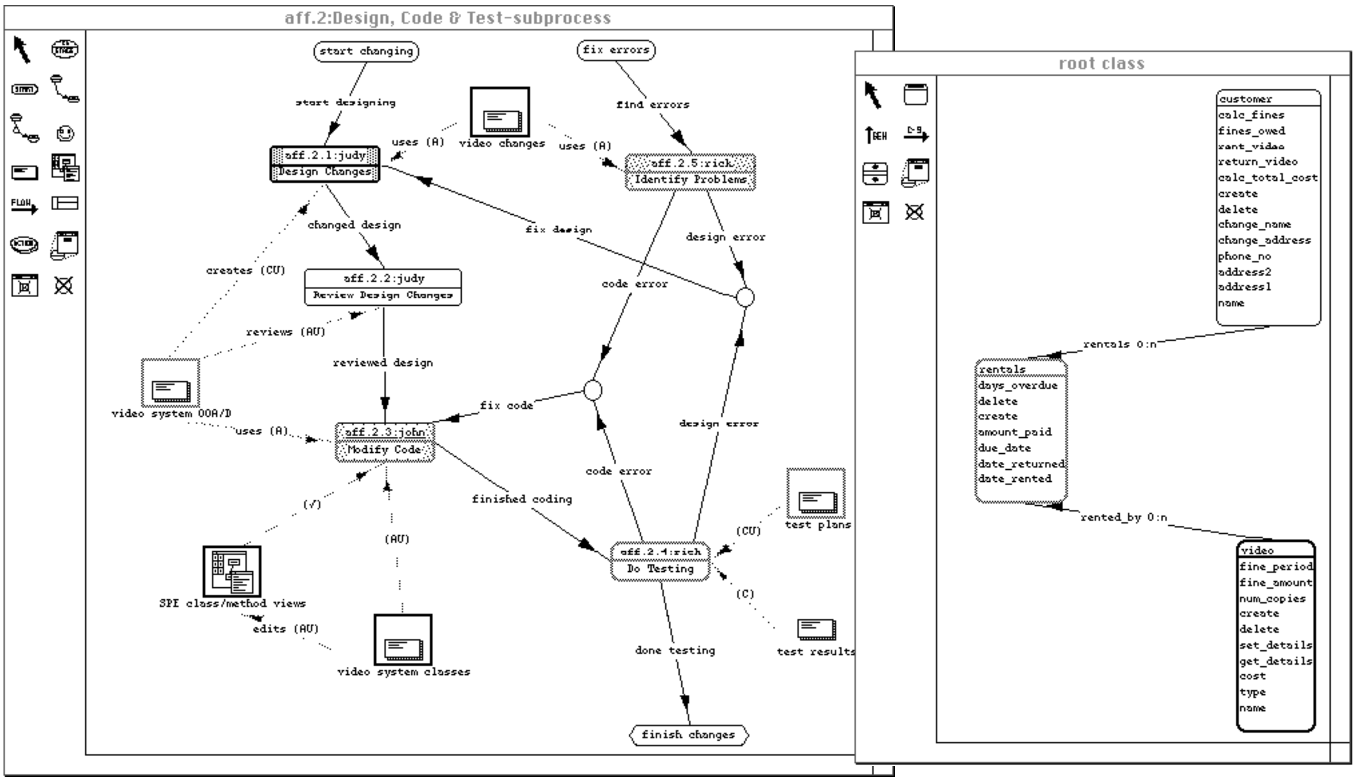


Figure 6. Highlighting Serendipity and SPE view icons to support group awareness.

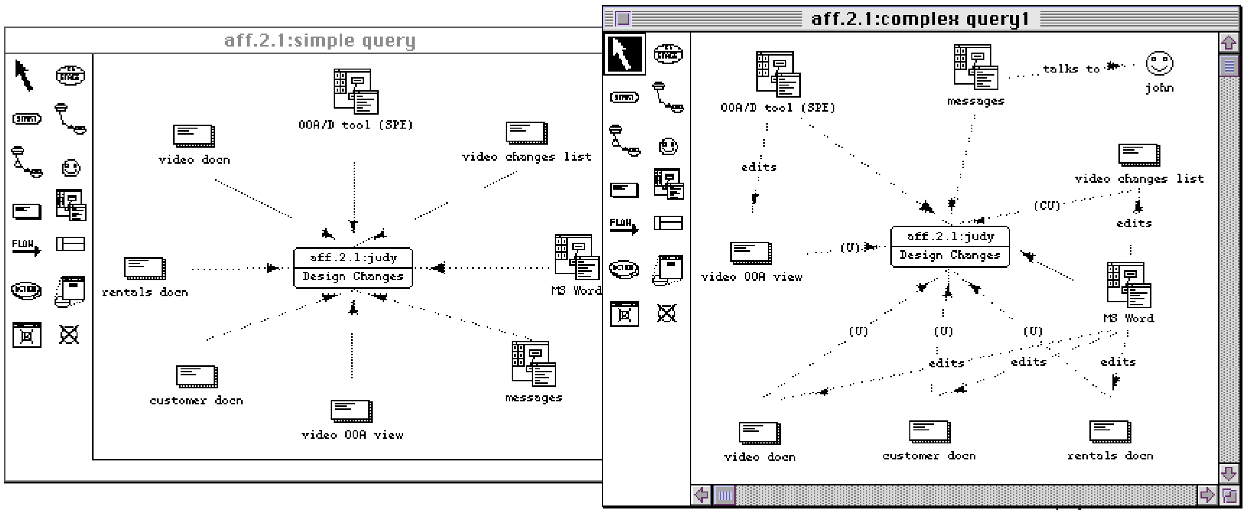


Figure 7. Visualisations of determined work context information for a process model stage.

Additional filtering can be applied to restrict a visualisation to show specific artefacts, tools and people, more or less frequently-used items, artefacts and tools used within a particular time range (eg. only recently-used vs. older ones). It is also possible to highlight determined work context information in various additional ways (eg. most frequently used artefacts/tools larger or in a different colour, recently used tools and artefacts coloured or shaded differently, etc.). The right-hand visualisation in Figure 7 is showing additional information to the simple query on the left. Views showing the overlapping work contexts for stages, where the items common to the determined work contexts of stages are highlighted, can also be generated.

6. Automatic Process Enactment

In Serendipity, developers are allowed to have several process model stages enacted at one time, but only one of these may be the “current enacted stage” ie. the stage for which work is currently being done. Very often developers switch between these enacted stages doing small amounts of work on several of them e.g. modifying a design, implementing part of the code, testing the partially–modified part of the system.

We can utilise determined work context information to “automatically” change the current enacted process model stage for a developer, if this is an appropriate thing to do. This allows developers to avoid having to manually change their current enacted process model stage when an obvious change in their work context occurs. For example, a developer may have the “design changes” stage as their current enacted stage, but also have the “code changes” and “test changes” stages enacted i.e. the developer is switching between these three enacted stages doing some work on each, rather than doing them in “waterfall” sequential fashion. If the developer starts to edit a test plan, their determined work context for “design changes” will not contain this artefact, but “test changes” will, indicating the developer is now working on “test changes”, which should now be made their current enacted stage.

To achieve a form of automatic switching of current enacted stage, we determine if a developer is now using a tool or artefact, or communicating with another user, not before determined as part the work context for their current enacted process model stage. An illustration of how this works is shown in Figure 8. All developer-generated events are monitored and the filter/action attempts to determine an appropriate alternative process stage from the recent work history of the developer, and if one is judged suitable, then it makes that stage the current enacted stage for the developer. Obviously this assumption of a context switch may be in error, so developers usually choose to use this facility only if they have built up determined work context information and a personal work history over a period of time for the process models they are using.

We now allow Serendipity to have no process model stage selected as the “current enacted stage” for a particular developer ie. no specified stage for which it is indicated current work is being done. This gives developers more freedom to do “exploratory” work without having to specify a particular process model stage is being worked on. We use a developer’s work history to record events generated while in this state. When the developer subsequently specifies their current enacted stage, the events cached while they had no current enacted stage can be utilised to process and record these events “after the fact”. This use of a determined work history supports much more flexible use of evolving process models than most existing workflow and process modelling environments.

When a developer changes their current enacted process model stage (or this is done automatically as described above), their work context changes ie. the artefacts, tools and collaborators they are interested in and/or affected by change. When the current enacted process model stage changes, highlighting needs to be changed in both Serendipity and SPE views. The determined work context for the newly enacted process model stage and the work history for the developer are used to achieve highlighting appropriate to a new work context.

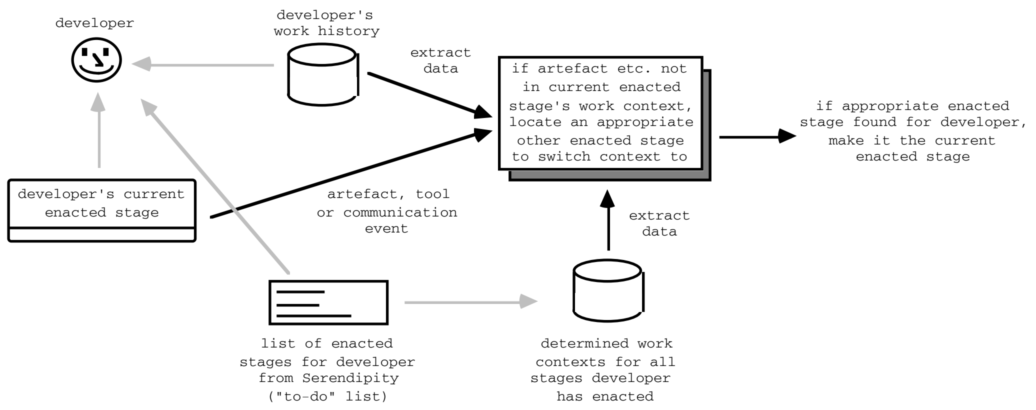

developer
developer's work history
extract data
if artefact etc. not in current enacted stage's work context, locate an appropriate other enacted stage to switch context to
if appropriate enacted stage found for developer, make it the current enacted stage
developer's current enacted stage
artefact, tool or communication event
extract data
list of enacted stages for developer from Serendipity ("to-do" list)
determined work contexts for all stages developer has enacted

Figure 8. An example of automatic change of enacted process stage based on change in work context.

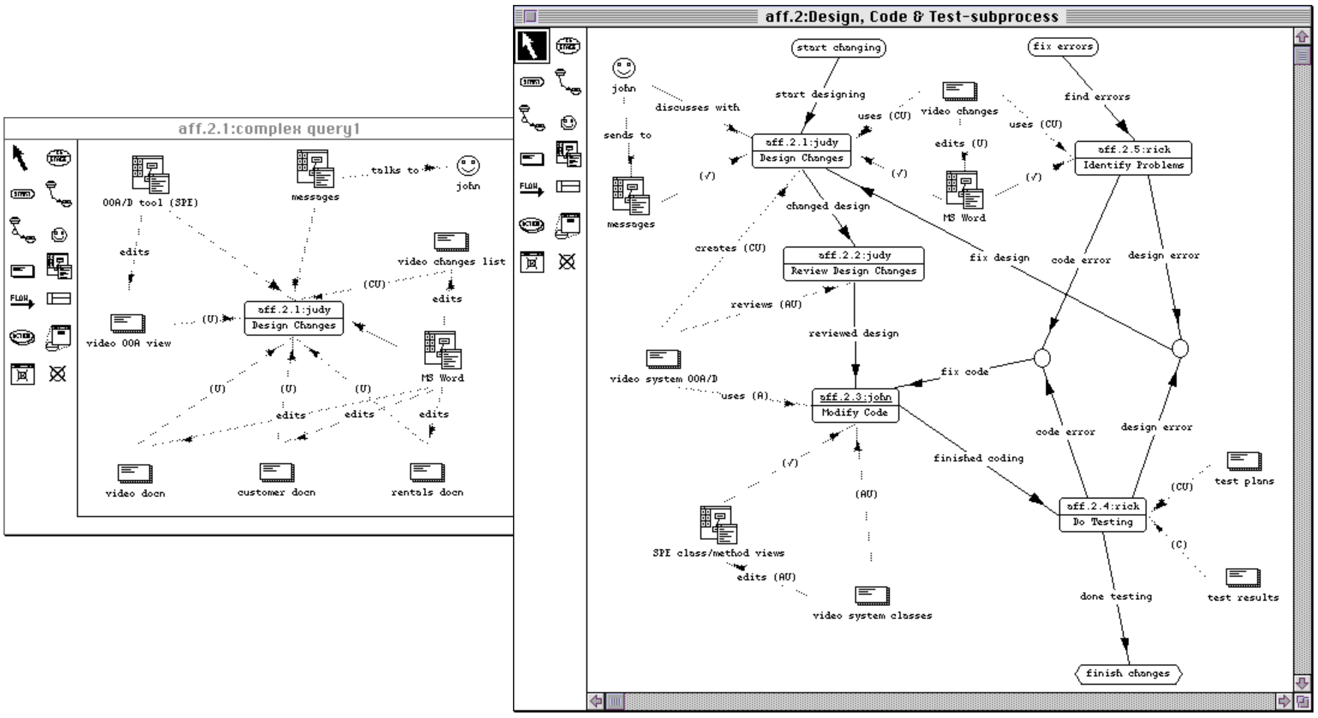


Figure 9. Abstracting and codifying visualised work contexts into reusable template process models.

7. Software Process Improvement

Software process model improvement can also utilise our work context determination and visualisation techniques. As developers have a collated record of work history for process model stages (and other developers), this can be used to refine software process models codified in Serendipity.

For example, Figure 7 showed a generated visualisation of the work context for stage aff.2.1., showing the software artefacts, tools and collaborators utilised for aff.2.1. on an actual development project. Obviously the artefacts and collaborators are project-specific, but a software process modeller can abstract a codified (ie. defined work context specification) from such visualisations, as illustrated in Figure 9. The visualisation on the left, generated from a stage's determined work context, has been used by a developer to refine the codified work context for process model stage aff.2.1., which now takes into account the communication tool, other developer and document editing tool shown in the determined work context visualisation. We have found the determined work context information for a stage to be very useful for restructuring and refining process models.

8. Implementation and Experience

SPE and Serendipity were developed by reusing the MViews framework for constructing ISDEs [17]. MViews provides a general model for defining software system data structures and tool views, with a flexible mechanism for propagating changes between software components, views and tools. ISDE data is described by *components* with *attributes*, linked by a variety of *relationships*. When a component is updated, a *change description* is generated. Change descriptions are propagated to all components dependent upon the updated component's state. Dependents interpret these change descriptions and possibly modify their own state, producing further change descriptions. This change description propagation mechanism supports a diverse range of software development environment facilities, including attribute recalculation, multiple views with flexible, bi-directional textual and graphical view consistency, a generic undo/redo mechanism, component versioning, and collaborative view editing [14].

SPE and Serendipity have been integrated by modifying MViews so that it sends change descriptions generated by tools to Serendipity, and by extending Serendipity to handle these artefact update and tool events. Relationships between the Serendipity base view and the SPE base view translate events (in the form of change descriptions) from one environment into appropriate events in the other.

Rather than alter the implementations of Serendipity and SPE directly to provide the work context and work history determination, visualisation and utilisation techniques described in the previous Sections, we have used Serendipity's filter/action language to implement them. This has the advantage of not having to modify the source code of the environments, allows developers to look at how the various determination, visualisation etc. utilities have been implemented, and because of the high-level nature of our filter/action language, this allows developers to tailor these to their own needs.

A simple work history determining filter/action is shown in Figure 10. Artefact, tool and communication events generated by the developer's actions are stored in a work history artefact (right-hand event flow). Any "set current stage" enactment events cause this work history to be partitioned, grouping artefact etc. events according to which stage the developer currently had enacted. Process model users can modify this filter/action to only record particular kinds of events, or to record them in different work history artefacts.

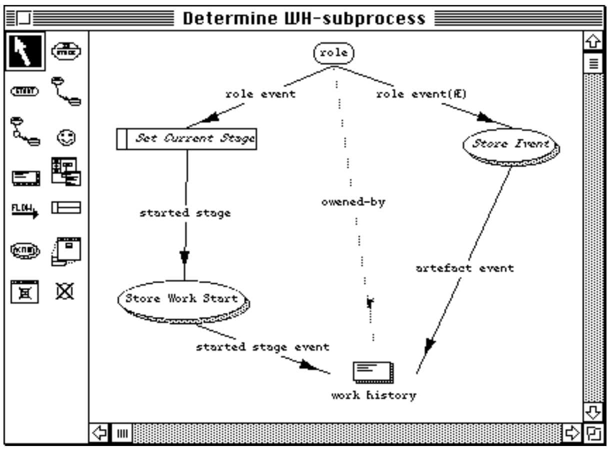


Figure 10. Filter/action determining work histories.

This approach of using the built-in Serendipity filter/action language has the disadvantage of limiting how these work context and history determination techniques are used. Developers must attach an icon representing the overall filter/action model to developer or stage icons in other views, which is not always an ideal approach. In addition, the interpreted nature of these filter/action models means their execution is currently quite slow e.g. half a minute to generate a work context visualisation like in Figure 7.

We have used our event history compilation, visualisation and automatic enactment mechanisms on several small software process problems. Our techniques have proven to be useful for allowing developers to effectively visualise past, current and likely future work contexts of themselves and other developers. They are also very helpful in assisting process improvement and alleviating manual context switching between multiple enacted stages.

Due to the slowness of the current SPE-Serendipity environment, and its limited integration with third party tools, we have built a new Java Beans-based prototype environment, JComposer. This provides much more efficient filter/actions and a much more open architecture. We are planning to port Serendipity to this new architecture and reimplement the techniques described in this paper. An industrial evaluation of the techniques on larger software development projects should then be possible.

9. Summary

When working on large, multi-user software development projects, developers need assistance in maintaining awareness of others' actions and in using and improving their software process models. We have described several techniques that record enactment and artefact change events generated during a development project. We then use these event histories to help maintain awareness of actions between developers, to enable an environment to automatically switch work context for a developer when appropriate, and to help developers improve codified software process models.

We are currently working on improving the analysis techniques used to examine work histories for developers, in order to deduce patterns of artefact and tool usage. This will allow us to more accurately predict usage patterns of these artefacts and tools, thus improving the visualisations of predicted work contexts for other developers. In addition, the merging of determined work context information from multiple projects promises to yield more accurate visualisations of work context changes over time, once again helping to keep developers better informed of others' work and to assist in process improvement. We are porting our process-centred environment to a new Java platform to make its architecture more open and improve its speed of operation.

References

[1] Baldi, M., Gai, S., Jaccheri, M.L., and Lago, P. Object Oriented Software Process Design in E^3, in *Software Process Modelling & Technology*, Finkelstein, A., Kramer, J. and Neusebeh, B. Eds, Research Studies Press, (1994), pp. 279-292.

[2] Bandinelli, S., Fuggetta, A., and Ghezzi, C. Process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, vol. 19, no. 12 (December 1993), 1128-1144.

[3] Bandinelli, S., Fuggetta, A., Ghezzi, C., and Lavazza, L. SPADE: an environment for software process analysis, design and enactment, in *Software Process Modelling & Technology*, Finkelstein, A., Kramer, J. and Neusebeh, B. Eds, Research Studies Press, (1994).

[4] Belkhatir, N., Estublier, J., and Melo, W.L. The Adele/Tempo Experience, in *Software Process Modelling & Technology*, Finkelstein, A., Kramer, J. and Neusebeh, B. Eds, Research Studies Press, (1994).

[5] Ben-Shaul, I.Z. and Kaiser, G.E. A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment. In *Sixteenth International Conference on Software Engineering,* IEEE CS Press, May 1994, pp. 179-188.

[6] Bogia, D.P. and Kaplan, S.M. Flexibility and Control for Dynamic Workflows in the wOrlds Environment. In *Proceedings of the Conference on Organisational Computing Systems,* ACM Press, Milpitas, CA, November 1995.

[7] Conradi, R., Hagaseth, M., Larsen, J., Nguyen, M.N., Munch, B.P., Westby, P.H., Zhu, W., Liu, C. EPOS: Object Oriented Coopeartive Process Modeling, in *Software Process Modelling & Technology*, Finkelstein, A., Kramer, J. and Neusebeh, B. Eds, Research Studies Press, (1994), pp. 33-70.

[8] Fernström, C. ProcessWEAVER: Adding process support to UNIX. In *2nd International Conference on the Software Process: Continuous Software Process Improvement,* IEEE CS Press, Berlin, Germany, February 1993, pp. 12-26.

[9] Fitzpatrick, G., Tolone, W.J., and Kaplan, S.M. Work, Locales and Distributed Social Worlds. In *4th European Conference on Computer-Supported Cooperative Work,* Kluwer Academic Publishers, Stockholm, Sweden, 1995.

[10] Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B. Connecting the pieces, in *Visual Object-Oriented Programming,* Burnett, M., Goldberg, A. and Lewis, T. Eds, Manning/Prentice-Hall (1995).

[11] Grundy, J.C., Hosking, J.G., and Mugridge, W.B. Towards a Unified Event-based Software Architecture. In *Joint Proceedings of the SIGSOFT'96 Workshops,* ACM Press, San Francisco, October 14-15 1996, pp. 121-125.

[12] Grundy, J.C., Hosking, J.G., and Mugridge, W.B. Low-level and high-level CSCW in the Serendipity process modelling environment. In *Proceedings of OZCHI'96,* IEEE CS Press, Hamilton, New Zealand, Nov 24-27 1996.

[13] Grundy, J.C., Hosking, J.G., Mugridge, W.B., and Amor, R.W. Support for Constructing Environments with Multiple Views. In *Joint Proceedings of the SIGSOFT'96 Workshops,* ACM Press, San Francisco, October 14-15 1996, pp. 212-216.

[14] Grundy, J.C., Hosking, J.G., and Mugridge, W.B. Supporting flexible consistency management via discrete change description propagation. *Software - Practice & Experience*, vol. 26, no. 9 (September 1996), 1053-1083.

[15] Grundy, J.C., Venable, J.R., Hosking, J.G., and Mugridge, W.B. Coordinating collaborative work in an integrated Information Systems engineering environment. In *Proceedings of the 7th Workshop on the Next Generation of CASE tools,* Crete, 20-21 May 1996.

[16] Grundy, J.C. and Hosking, J.G., Serendipity: integrated environment support for process modelling, enactment and improvement, to appear in *Automated Software Engineering: Special Issue on Process Technology*, Kluwer Academic Publishers, vol. 5, no. 1, January 1998 (in press).

[17] Grundy, J.C. and Hosking, J.G. Constructing Integrated Software Development Environments with MViews, *International Journal of Applied Software Technology* , vol. 2, no. 3-4, 1996.

[18] Hill, R.D., Brinck, T., Rohall, S.L., Patterson, J.F., and Wilner, W. The Rendezvous Architecture and Language for Constructing Multi-User Applications. *ACM Transactions on Computer-Human Interaction* vol. 1, no. 2 (June 1994), 81-125.

[19] Kaplan, S.M., Fitzpatrick, G., Mansfield, T., and Tolone, W.J. Shooting into Orbit. In *Proceedings of Oz-CSCW'96,* DSTC Technical Workshop Series, University of Queensland, Brisbane, Australia, August 1996, pp. 38-48.

[20] Kellner, M.I., Feiler, P.H., Finkelstein, A., Katayama, T., Osterweil, L.J., Penedo, M.H., and Rombach, H.D. Software Process Modelling Example Problem. In *Proceedings of the 6th International Software Process Workshop,* (Ed), T.K., IEEE CS Press, Hokkaido, Japan, 28-31 October 1990.

[21] Krant, R.E. and Streeter, L.A. Coordination in Software Development. *Communications of the ACM* vol. 38, no. 3 (March 1995), 69-81.

[22] Medina-Mora, R., Winograd, T., Flores, R., and Flores, F. The Action Workflow Approach to Workflow Management Technology. In *Proceedings of CSCW'92,* ACM Press, 1992, pp. 281-288.

[23] Di Nitto, E. and Fuggetta, A. Integrating process technology and CSCW. In *Proceedings of IV European Workshop on Software Process Technology,* LNCS, Springer-Verlag, Leiden, The Nederlands, April 1995.

[24] Ratcliffe, M., Wang, C., Gautier, R.J., and Whittle, B.R. Dora - a structure oriented environment generator. *IEE Software Engineering Journal*, vol. 7, no. 3 (1992), 184-190.

[25] Reiss, S.P. Connecting Tools Using Message Passing in the Field Environment. *IEEE Software*, vol. 7, no. 7 (July 1990), 57-66.

[26] Roseman, M. and Greenberg, S. Building Real Time Groupware with GroupKit, A Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 1 (March 1996), 1-37.

[27] Schmidt, K. and Bannon, L. Taking CSCW seriously: Supporting Articulation Work. *Computer Supported Cooperative Work (CSCW): An International Journal*, vol. 1, no. 1-2 (1992), 7-40.

[28] Swenson, K.D., Maxwell, R.J., Matsumoto, T., Saghari, B., and Irwin, K. A Business Process Environment Supporting Collaborative Planning. *Journal of Collaborative Computing*, vol. 1, no. 1 (1994).

[29] TeamWARE, I., *TeamWARE Flow*, 1996. (http://www.teamware.us.com/products/flow/).

[30] Wasserman, A.I. and Pircher, P.A. A Graphical, Extensible, Integrated Environment for Software Development. *SIGPLAN Notices*, vol. 22, no. 1 (January 1987), 131-142.