Tool Integration in Software Engineering Environments

M.N. Wicks
Department of Computer Science,
School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh, EH14 4AS, UK
Tel: +44 (0)131 451 3221
mnw1@macs.hw.ac.uk

ABSTRACT
This article presents doctoral research on tool integration within software engineering environments. Tool integration concerns the techniques used to form coalitions of tools that provide an environment supporting some, or all, activities within the software engineering process. Some interesting phenomena have been observed, such as the ad hoc nature of tool integration in one particular software engineering company. This observation is at variance to the common perception of widespread integration suggested by tool vendors and some previous academic literature. Initial results suggest that integration must be implemented for business reasons, not for its own sake.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques - *Computer-aided software engineering (CASE)*; D.2.6 [**Software Engineering**]: Programming Environments - *Integrated environments*; D.2.9 [**Software Engineering**]: Management - *Life cycle; Software process models*;

General Terms
Management, Measurement, Economics.

Keywords
Software Engineering, Software Engineering Environments, Software Tools, Tool Integration.

1. INTRODUCTION
The practical application of software development methods through the use of appropriate tools is often somewhat different to the original overall theoretical intention of the method. The integration of tools within a software engineering organisation is no different, as the practical realisation is often orthogonal to the suggested theoretical panacea. For instance, a desire to increase productivity to match ever increasing competition, curtails the freedom of an organization to choose its own style, method and pace of software development. Therefore the software process, and hence its supporting coalitions of tools, must be suitably modified and managed to maintain business advantage, and yet preserve the unique selling point of that organisation's products.

This paper briefly discusses a research effort that aims to discover more about the nature of tools used within software engineering environments, and how best they should be integrated to maximize possible productivity and quality gains for the organization. First the most significant prior research works are discussed to describe the state of the art. Then the main questions that this particular research effort is attempting to address are discussed. Following on from these hypotheses, the choice of method for this work is discussed. Next the initial outputs and data collected from the chosen method are presented. A brief critique of the chosen method is discussed together with some initial results. Finally, the key initial findings are recapitulated, and are presented together with a plan for the future work required to complete a thesis.

2. THE STATE OF THE ART
Tool integration has been an active research area for almost two decades, as described in [7]. Two seminal works that demand particular discussion are Wasserman [6] and Earl [3]. Earl suggested the "Toaster" Reference model for CASE Tool environments, whilst Wasserman proposed five dimensions that could be used to measure tool integration.

Earl's model put forward a common environment through which tools would be used. Common front-end services would be combined with back end data management facilities mediated by a messaging network, into which tools could be slotted; like slices of bread in a toaster. This infrastructure required all tools to be compliant with the framework by building them to a common standard. However, this suggested solution also precluded the use of tools outside of the framework.

Wasserman proposed a method for measuring the commonality between environments. He suggested five degrees, or "dimensions", that could provide a scale against which comparisons could be made. These dimensions are: platform integration describing the physical environment that the system runs on; presentation integration describing the user interface metaphor employed; data integration concerning the level of sophistication of the shared data structures; control integration concerned with the level of interoperability between the tools; and finally process integration that describes the level to which a particular software methodology can be reified.

All subsequent works in this area are built on these two seminal contributions, with a tendency for researchers to classify and clarify the levels of integration in candidate environments, for example, see the work detailed in [1], [2], [4], [5] and [8]. A thorough review of previous research has already been carried out [7], and this analysis highlights several interesting ideas about tool integration in software engineering environments. This review also shows that that

following a burst of activity in the early 1990s, work tailed off somewhat, only to re-emerge as a significant research topic in recent years. This review also highlights a worrying tendency where these more recent efforts do not consider the seminal efforts of Wasserman, Earl and others; so much so that there is a danger that there will be no new lessons and insights, with work merely being repeated.

A new research agenda for tool integration in software engineering environments is proposed; one that concentrates on the reasons for considering any integrated solution within an organisation. Among the key issues that are identified is the need to combat the urge to compare tools using their features only, for which the term "feature-itis" can be used. The contemporary work that has been reviewed, shows that efforts are continuing to create yet more solutions as per Earl, rather than following the investigative path that Wasserman took. The review [7] suggests that experience, be it organisational or personal, is a paramount factor in selecting tools and integrating them, and that investigations should be conducted into the motivation for integration rather than just propounding yet another architecture or model. Contemporary research into tool integration has so far avoided business realities when software engineering environments are created. For example, there is a need to focus on key issues to any organisation such as productivity gains and Return On Investment (ROI) that an integrated solution should provide.

3. HYPOTHESES

From the review [7] and its conclusions outlined above, a number of hypotheses emerge: the first hypothesis considers the relationship between the sustainability and the sophistication of any integrated tool solution, as described in Figure 1; the second hypothesis considers the relationship between the net value of an integrated tool solution, against the sophistication of that solution, as described in Figure 2; the final hypothesis considers the relationship between the sophistication of an integrated tool solution and the time taken to reach such a level of sophistication, as described in Figure 3.

The hypothesis demonstrated by Figure 1 contends that simple, focused integration solutions last longer than more complicated ones, perhaps since increasingly complex solutions run the risk of early obsolescence. The implications of this suggestion are of interest to those planning investment in tools and services to develop software, as it could suggest that expensive, sophisticated solutions are to be avoided. As the sophistication of a tool integration solution increases, there is an implication that more activities within the software lifecycle would be covered by the resulting solution. So if few activities are integrated into a solution, will this then last longer (and so provide a better ROI) than further attempts to integrate yet more activities?

The hypothesis demonstrated by Figure 2 suggests that there is an optimum derived "value" that can be achieved by using a particular integrated set of tools. This also has implications for the planning of investments in new or improved tool coalitions, as it suggests that there may be a level of investment beyond which there would be little point in investments to increase the sophistication of the proposed tool coalition. Again this might also suggest that there may be differing levels of sophistication of tool integration between differing software activities within the software lifecycle, and that these differing combinations of sophistication may result in an optimal return, be it in terms of productivity or quality to the organization.

The final hypothesis demonstrated by Figure 3 contends that there is a relationship between the sophistication of integration, used in any tool coalition, and time. Simply put, the level of integration sophistication between tools increases over time, as the utilising organization gains experience of a set of tools, and learns how best to adapt them to match their own particular business processes, models and ethics.

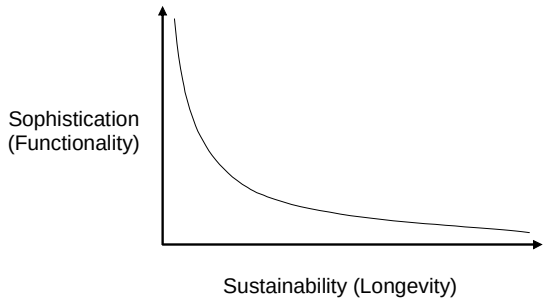

Figure 1 Sophistication vs. Sustainability

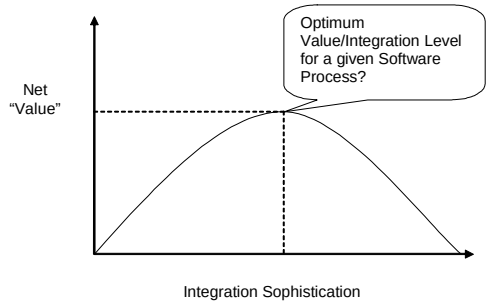

Figure 2 Net "Value" vs. Sophistication

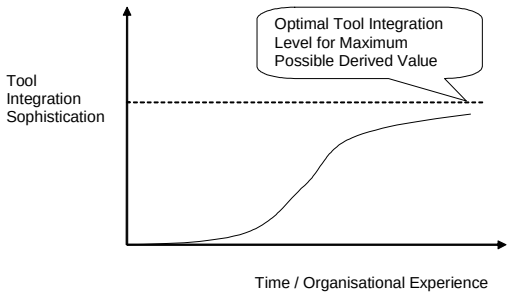

Figure 3 Sophistication over Time

4. APPROACH

In order to explore these wide ranging hypotheses, a collaboration was formed between Heriot-Watt University and a software company in Scotland that produces a single integrated suite of programs for the telecommunications sector. The collaboration coincided with the company embarking on a process improvement programme with the goal of accruing a significant productivity benefit. Initially the company wanted to establish a baseline from which subsequent productivity gains could be identified and measured; in particular, those accruing from the adoption of the Rational suite of tools, a strategic decision by the company, soon to be made available to all development teams. At this stage the company wants to remain anonymous until results of significance emerge. The company are aiming for a 25% productivity improvement over a two year period, which they hoped would accrue from adoption the Rational suite.

The work started by holding a series of semi-structured interviews with representatives from all the development teams in the company. During preliminary planning meetings with the champion of the Rational project, the design of the interview was significantly tailored to the culture and terminology of the collaborating organisation, in order to remove multiple explanations of the same term in every interview, for example, the names of the lifecycle phases used in a project or single block of work (as shown across the X axis in Figure 4, being: Project Management; Requirements Management; Analysis & Design; Implementation; Testing; Defect Tracking & Change Control; and finally Configuration Management).

The interview comprised three sections; general, software process and tool integration characteristics. General team characteristics were collected first. These included individual and collective roles and responsibilities, team size and composition (numbers of project managers, architects, developers and testers), granularity and timing of the planned work, programming languages used, the proportion of time devoted to maintenance and to development work, as well as the teams' methodological approaches. Next the formality of the software process for each lifecycle activity was established, together with the specific tools used, whether any metrics were collected or used, whether any tools were integrated, a subjective statement of effectiveness of the team at each activity and a statement of the effectiveness of the tools used. Finally, the interviewees were asked to identify good examples of tool integration they knew of in the organisation, and to identify any areas that would benefit from integrated tools that are poorly supported at present.

Eleven interviews were conducted with fourteen people attending, covering eleven teams responsible for a total of seventeen projects within the company. Each interview lasted no more than an hour without any tape recording at the request of the company, so only notes were taken.

5. PRELIMINARY RESULTS

The interview notes were analysed and, where possible, transformed into graphical formats. For instance, Figure 4 illustrates the frequency of tool integration within each lifecycle activity. "Yes" indicates that fully automated integration exists within this activity, "No" indicates no integration whatsoever, "Some" indicates informal integration only, and "Not Applicable" means that the activity is not performed by that team. In this case, the company had tools to address both defect tracking and configuration management that were very well integrated, thus enabling releases of software to be created that included all the required fixes to identified defects. In other words, one tool integration instance spanned more than one lifecycle activity; reflected in the two largest "Yes" peaks that can be seen in Figure 4.

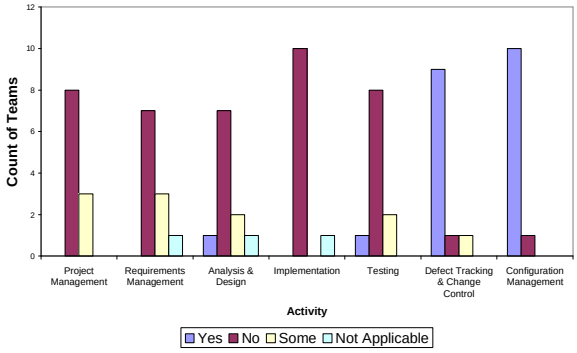

Figure 4 Tool Integration across Lifecycle Activities

From Figure 4, it can be seen that integration is not uniform across the lifecycle. The degree of fully automatic tool integration has only been achieved between two activities, with a few instances of manual integration elsewhere, suggesting that automatic integration may be a rare phenomenon in a practical industrial setting. By manual integration we mean the use of "cut and paste" between commodity software applications, for example those within Microsoft Office.

6. EVALUATION

These initial results provide evidence that integration is not uniform across the whole of an organisation's software process, such that it could be said that only "Islands of Integration" exist. Not only this, but the whole software process is not uniformly defined equally well across all lifecycle activities. Fully automatic tool integration only occurs when a decision is taken to reify an agreed and defined software process. In this instance, the integration is between the Defect Tracking & Change Control and the Configuration Management activities, and this was driven by a perceived inability to deliver to customers the required software, containing only those components relevant to a specific release. Thus, the driver was the desire to satisfy a specific, customer facing business goal. This result was not initially suggested by any of the previously mentioned hypothesis, but it does emphasise the importance of business factors to the selection or adoption of a particular level of tool integration sophistication.

At the moment all of these initial results are contextual and subjective, so support from further objective measures is required. These objective measures need to describe project characteristics such as size, longevity, or complexity, as well as describing staff skill levels. Therefore, no definitive conclusions can yet be reached, and work is ongoing. However, these useful insights provide encouragement for

further investigations into the hypotheses suggested previously in Figures 1, 2 and 3, because the observed tool integration phenomenon has been realised by a simple tightly defined process exercised by automatic integration. Our future work will include conducting a similar exercise with another company to provide corroboration of these initial findings, and to repeat the same exercise in a years time with the original company in order to measure the effect of the introduction of the Rational suite across all teams. This future exercise will also provide a useful temporal dimension to this data.

Also of note here are questions relating to the appropriateness of the method chosen. This raised a number of issues once the interviews were under way, as it soon became apparent that not every question was going to have an answer, so missing answers often resulted. For example, if a team collected no metrics within a particular activity, then the follow up question that attempts to determine when the collection of metrics started is not asked. Similarly, it also became apparent that not all questions were relevant to everybody, so whole sections could sometimes be omitted. For example, asking the manager of the maintenance team that solely implements bug fixes about his Analysis and Design process was fairly pointless. This suggests that the design of any interview process must be sufficiently flexible to cope with a wide enough range of possible answers, and yet maintain sufficient focus within the particular set of constraints posed by the organisation under investigation. This will have knock on consequences, when the investigation is repeated with other organisations.

7. CONCLUSIONS

In this summary paper, research into tool integration has been presented and discussed, with particular reference to a single software engineering company. Some background motivations to our work have been discussed, and these suggest that researchers are in danger of falling into the trap of answering old questions, which have little to do with current business concerns or the practical application of tools to produce software. These new questions must relate to business priorities and goals, and not merely to exercise technical issues. To this end, business is increasingly aware of the investment required to support the development of software, such that the need to justify the time and money required to create sophisticated software engineering environments is paramount, as well as needing to calculate some indication of when this investment will start generating a return.

A series of semi-structured interviews has been conducted across a single company, from which an initial set of results has been produced. From these results, it can be seen that tool integration is not a uniform phenomena across the software lifecycle, instead “Islands of Integration” exist. These “Islands” have been created to answer a customer-facing deficiency, and not a theoretical need to satisfy some abstract software standard, process model or technical issue.

This paper demonstrates the surprising observation that at least one successful company is not concerned with perceived best industrial practice, as exercised through a sophisticated set of tools integrated throughout the development lifecycle, but is rather more concerned with addressing immediate pragmatic imperatives.

The rest of the work required to produce a thesis ready for submission will start with an industry-wide online survey in Scotland. Comparative data will be extracted in a similar fashion from at least two other organizations, together with a return to the original organization after one year to provide a temporal dimension. The goal is then to draft a predictive model, or at least the first steps towards such a model, that will assist organisations in their selection of tools and how best to create coalitions that optimize their work methods, processes and ethics.

8. ACKNOWLEDGEMENTS

M N Wicks is financially supported by Heriot-Watt University throughout the course of his studies for his Doctorate in Computer Science.

9. REFERENCES

[1] Amsden, J. Levels of Integration. Object Technology International, 2001. (http://www.eclipse.org/articles/Article-Levels-Of-Integration/Levels Of Integration.html), [last accessed 11/3/05].

[2] Brown, A.W. Control integration through message-passing in a software development environment. *Software Engineering Journal 8(3)*: 1993, 121-131.

[3] Earl, A. Principles of a reference model for computer aided software engineering environments. In *Proceedings of The International Workshop on Environments (Software Engineering Environments)*, LNCS 647, Springer, 1989, 115-129.

[4] Rader, J., Morris, E.J. and Brown, A.W. An investigation into the state-of-the-practice of CASE tool integration. In *Proceedings of the Software Engineering Environments Conference*, 1993, 209-221.

[5] Thomas, I. and Nejmeh, B. A. Definitions of tool integration for environments. *IEEE Software 9(2)*, 1992, 29-35.

[6] Wasserman, A.I. Tool integration in software engineering environments. In *Proceedings of the International Workshop on Environments (Software Engineering Environments)*, LNCS 647, Springer, 1989, 137-149.

[7] Wicks, M.N. *Tool integration in software engineering: The state of the art in 2004.* Technical Report HW-MACS-TR-0021, Heriot-Watt Institute of Software Engineering, 2004.

[8] Yang, Y. and Han, J. Classification of and experimentation on tool interfacing in software development environments. In *Proceedings of the (Asia-Pacific) Software Engineering Conference*, 1996, 56-65.