Using the Event-B Formal Method for Disciplined Agile Delivery of Safety-critical Systems

Andrew Edmunds*, Marta Olszewska† and Marina Waldén‡
Distributed Systems Lab.
Abo Akademi University,
Turku, Finland
Email: *aedmunds@abo.fi, †mplaska@abo.fi, ‡mwalden@abo.fi

***Abstract*—In order to improve the development process of high-integrity systems, using formal methods, we consider how agile techniques may influence the Event-B formal method, and how Event-B may be used in a development that uses an agile approach. To examine the crossover between Event-B and agile methods we review the Disciplined Agile Delivery approach (DAD). The DAD approach is inspired by many state-of-the-art agile techniques, and we use it as a meta-analysis of current best-practice. In this paper, we propose an agile process for using Event-B and examine how agile techniques might influence the use of Event-B. We identify a number of areas in which Event-B could be improved and suggest that a different view of agile practices may be needed for an agile project involving formal development.**

Keywords–Agile; Formal Methods; Event-B; Critical Systems

I. INTRODUCTION

As part of the ADVICeS project [1], we have been investigating the crossover between agile and formal methods [2], [3]. In particular, we are focussing on the Event-B method [4] and DAD [5]. Event-B is a formal approach for the rigorous specification of safety-critical systems, based on set-theory and predicate logic. In contrast, DAD is a pragmatic, flexible approach to agile development, which seeks to guide teams towards a solution, rather than prescribe a list of tasks that should be adhered to. DAD's creators are experienced agile practitioners, taking inspiration from the many agile methods they have encountered during their careers, including XP [6], Scrum [7] and Lean approaches [8]. Since the DAD approach is so wide ranging, we use it for a meta-analysis, to assess how DAD and Event-B could be used together. The result provides an insight into the most relevant issues. We believe that this may be of interest to others in the formal methods community, since the questions we raise, and seek to answer, may apply to other methods. In Section II, we introduce Event-B. In Section III, we look at DAD and how it relates to Event-B. In Section IV, we examine how Event-B features may be used in a DAD project. In Section V, we discuss process goals. In Section VI, we discuss related work, and in Section VII, we summarize and reflect, asking the question "What's stopping Event-B from being used in an agile development?"

II. EVENT-B

Event-B is a specification language and methodology with tool support called Rodin [4], [9], [10]. Event-B has received interest from industry, for the development of railway, automotive, and other safety-critical systems [11]. In Event-B, system properties are specified using set-theory and predicate logic; proof and refinement are used to show that the properties hold as the development proceeds. It is recommended that developers add detail to the specification in a series of refinement steps. Proof obligations are automatically generated by the Rodin tool; the automatic prover is able to discharge many of them. Event-B is designed to reduce the amount of interactive proof needed, during the specification and refinement steps [12]. Complex systems can be handled using methods of composition and decomposition [13].

context C0
sets S ...
constants
c ...
axioms
c ∈ S

Figure 1. Example Event-B Context

A. Event-B's Core Technology

The core Event-B modelling artefacts are Machines and Contexts. Contexts describe the static elements of a system, using sets, constants and axioms. Machines describe the dynamic elements of a system; a machine can *see* contexts, and is made of variables, invariants, and guarded events. A simple context is shown in Figure 1. The context C0 declares a carrier set *S*, and a constant *c*, which is typed in the axioms as a member of *S*. Axioms can be used to type constants and specify assumptions. There are a number of built-in types such as $\mathbb{Z}$ and BOOL, and new types and operators can be added [14].

In Figure 2, we have a machine $M0$, which *sees* the context $C0$. In this way, $M0$ gains access to the sets and constants declared within $C0$. The machine's variables are typed in the invariants clause, which may also contain the description of other system properties.

Events describe atomic state updates; they are non-deterministically scheduled, occurring only when the guards (in the *where* clause) are true. The example shows a parameter *p1*, which is typed as an integer. It has a guard, which states that the updates can only occur when $x < p1$. In the action (the

 ISBN: 978-1-61208-458-9

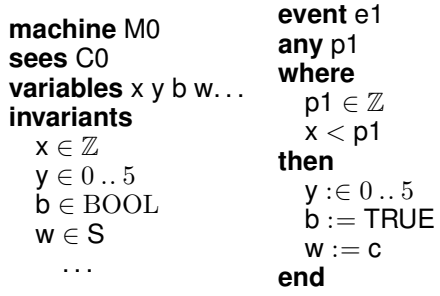

Figure 2. Example Event-B Machine

then clause) y is non-deterministically assigned a value in the range $0 \ldots 5$. Variables b and w are assigned (deterministic assignment) TRUE and c, respectively. All three assignment actions take place simultaneously, in parallel.

Refinement is used to add detail to a model. A refining machine is added to the development, then new variables and events can be added; or abstract events can be refined by strengthening the guards, and adding new actions. The new actions can modify the newly introduced variables.

B. Extensions to the Core Functionality

The Rodin tool is based on Eclipse [15], an open, extensible tool platform that allows plug-in developers to contribute new tools. There are many Rodin plug-ins available [16], we describe a few of them in this section.

One of the first steps in a project is to analyse the requirements. In Event-B, requirements can be recorded using **ProR** [17]. Elements in the Event-B model can be linked to references in the requirements for traceability throughout the development. Use case analysis can be done using a use-case modelling plug-in [18].

The **Theory** extension provides a way to add new data types, operators, inference and rewrite rules, and code generation translation rules [14], and theories can be reused in different projects. There are several extensions that provide **diagrammatic representations** with Event-B semantics, such as state-machines [19], class-diagrams [20], and progress diagrams [21].

When modelling, it is recommended that the behaviour of the model be explored using **ProB** [22], an animator and model-checker for Event-B. ProB allows developers to make sure that the expected behaviour, of a model, is observed. It can also be used to interrogate a model's state when proofs fail. ProB's model-checking facilities include a Linear Temporal Logic feature. An SMT solver is available in a separate plug-in [23]. Typically, ProB is used by technically-aware team members. There are additional animation tools for the non-technical stakeholders, to help them understand the system. Two such tools are **AnimB** [24] and **B-Motion Studio** [25]. Both perform similar roles, where the underlying elements of an Event-B model are linked to some graphical representations. As events occur, the state changes, together with the visual representation. This can be especially useful when validating requirements, and when showing the development to non-technical team members and stakeholders.

As models become more complex, **decomposition** techniques can be used to make the model of the system more tractable [13]. This also facilitates parallel development, since the sub-models can be refined independently. It can be used to introduce structure in the model, and ultimately, it provides a means to refine into implementation structures. The composed machine component provides the glue for linking the sub-models. This can also be used to compose independently produced components. Another approach is to use the **modularisation** plug-in [26], which takes a different view. It uses interfaces, and it introduces the concepts of operations, pre-conditions, and post-conditions into Event-B.

As the development progresses, it may be useful to perform simulations with a continuous representation of the environment, or with a continuous model of parts of the system. This may be achieved with **multi-simulation tools**, such as that based on the Functional Mock-up Interface (FMI), for simulation of cyber-physical systems [27]. This approach facilitates modelling, using a mix of continuous and discrete models, which can provide more confidence that non-functional requirements will be satisfied. A further step is to generate code from the Event-B models, which can then be included in an FMI simulation [28]. This simulation uses code that is very close to the actual software that will run in the system, and can easily be tailored for use in deployment. This approach is an extension of the work introduced in [29], where **code generation** tools use a scheduling language to describe implementations for multi-tasking, embedded systems. The target languages include Ada, Java and C. Another approach to automatic code generation includes that of Rivera et al. [30], who produce Java code, and Java Modelling Language (JML) annotations for downstream verification. Whether or not the code generator is certified, it can be desirable to perform testing. There is a **model-based testing** tool available for Event-B [31], which generates test-cases from the Event-B models based on ProB.

III. DAD

The Agile Manifesto [32] was presented in 2001, bringing together many new ideas that facilitate efficient, and timely, software development, in an ever changing environment. A number of concepts figured prominently. These include early delivery of "useful" resources; iterative development; collaboration between customers, and the development team; and collaboration within the development team itself. A number of approaches were advocated, such as XP [6], Scrum [7] and Lean [8]. These approaches consist of "guidelines for delivering high-quality software in an agile manner" [33]. Hence, it is apparent that agile has no single definition, and the approaches do not always cover all the necessary aspects of a development.

In [34], Ambler and Lines comment on the fact that agile practices were often adapted to the needs of the businesses in which they were applied. These authors advocate DAD [5], which provides a framework for this process. It includes a set of guidelines, drawn up as a result of analysing where agile approaches fall short. In the remainder of the paper, we make use of DAD, to understand how Event-B may be used in an agile project.

DAD is described by its authors as an adaptable process framework; they identify and discuss the many techniques and tools that may be used during agile development. Instead of building on radical, new technologies, the framework adopts

 ISBN: 978-1-61208-458-9

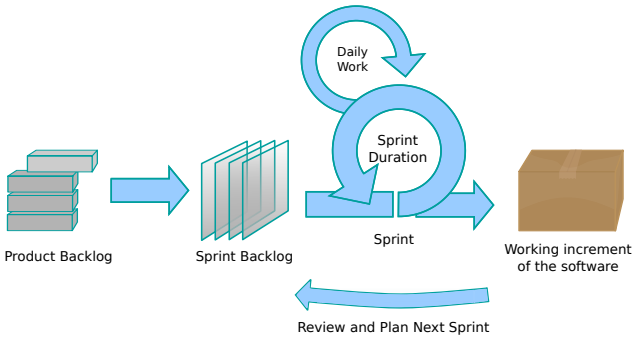

Figure 3. The Traditional Scrum Life-cycle

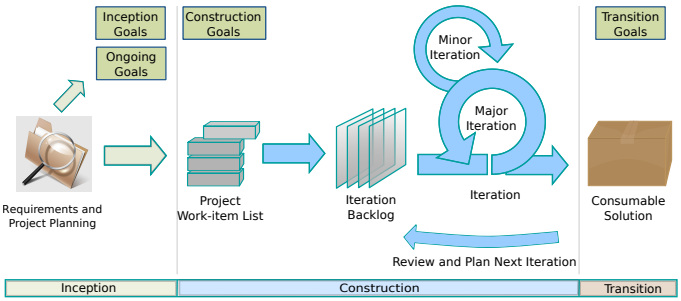

Figure 4. A Disciplined Agile Delivery Life-cycle

a pick-and-mix approach from existing technologies, based on the authors' practical experience in industry. Rather than being a step-by-step guide, there are a set of points to consider, and a set of guidelines that may be followed if appropriate. Many of the techniques that are part of DAD can also be used with Event-B. Much of the advice relates to project planning issues. One of the decisions includes the choice of which technologies to use, such as, whether to use Event-B. If the decision is yes, then a choice of modelling features is available; options include state-diagram [19] and class-diagram representations [20]. The scope of the formal model should also be defined.

1) Project management: Some of DAD's core concepts concern project management; others relate to individuals and team-working; and, others relate to tools and techniques. From the project management perspective, DAD does not impose a single life-cycle model. We understand it as a process goal-driven approach; as a side-note, the process goals in DAD should not be confused with goal-based requirements. In a process goal-driven approach, a semi-formal decision-making process is advocated, taking place at certain stages in the development, with the aid of process goal diagrams. The diagrams are tree-structured, and identify (at its root) a process decision point - with sub-branches showing the aspects (called process factors) to be considered. The leaves identify the choices that are available. We have identified this as one area which could be useful for Event-B, see Section V.

Scrum is considered to be a construction-centric approach focussed on producing working code at the end of each iteration, see Figure 3. With Scrum, code delivery is seen as the measure of progress. DAD can be viewed as an extension of Scrum, adding an inception phase, and a transition phase. It delivers consumable solutions, which can be seen as the more general notion of artefacts that are useful for stakeholders. In addition, DAD advocates the use of technology neutral terms; scrum's *sprints* are known as *iterations*, see Figure 4. We also note that there are goals for each development phase, and on-going goals that are considered throughout the project. Inception is the upfront consideration of design and architectural issues, and transition relates to the process of releasing a project.

In a safety-critical setting, there is likely to be more emphasis on gathering requirements early on, when compared to non-critical agile developments. Using DAD as a guide, it recommends that this stage is short relative to the construction phase of the project. For safety-critical parts of a system, this could involve a structured approach to requirements specification. We discuss how this might be done in Section V. But, the brevity is mitigated by the fact that the process will be iterative, so details evolve as the project progresses. In the DAD process, an important task of the inception phase is to scope the project and plan for the subsequent phases.

The activities performed during a DAD iteration are driven by a prioritised *work-item list*, which may contain requirements, change requests, feature requests, and so on. This expands on the scrum notion of a *product backlog* which may just contain requirements. The *sprint backlog* of scrum is known as an *iteration backlog* in DAD. The *work-item list* also takes into account the risks associated with the item. It is suggested that high-risk items are dealt with first, in order to minimise the risk to the project as a whole. It is recommended that Event-B is introduced to the process at an early stage of development, to begin modelling the critical parts of a system. Both approaches are compatible in this respect. Planning issues, such as which tasks should be undertaken during the next iteration, are discussed in the iteration planning phase. In safety-critical systems, we can choose to take a different view of *iterations*. When certification of the system is required, it is unlikely that this will take place more than three or four times a year, since it is a lengthy process. This means that there may be a different interpretation of iterations: *minor iterations* can be used for achieving and measuring project progress, with fully certified releases completed at the end of *major iterations*. The transition to a released, and supported product is of less interest in this paper. The life-cycle may be repeated from the inception phase, for each release of a product, including maintenance releases.

2) Team-working: DAD emphasises teamwork and the factors involving the individuals that make up the team. Motivated individuals are encouraged to learn, and make improvements, on an individual level, and at the team level. The lessons learned should be shared, too. New guidelines and patterns should be created where necessary. However, this needs to be done in a structured way, so that the patterns can be retrieved at an appropriate time in the future: there should be sufficient information available, to allow decision makers, and potential users, to make their choices. This can form part of the information that is available at the time of making process decisions, which we discuss in Section V.

3) Development Techniques: In the available literature, as in DAD, practitioners advocate the use of test-driven development. Test-driven development begins with the creation of tests that relate to some required feature. This creates a link between requirements, and tests, and provides traceability

 ISBN: 978-1-61208-458-9

between requirements and code. The test is first written so that it fails, then an implementation is coded so that the test no longer fails. The main advantage is that it ensures that the implementation passes the test, from a very early stage, and also, as features are introduced. However, there is no assurance that the test, itself, is correct. The test can be further used as part of a continuous integration process. "Test early and test often", is advice that is commonly given, and developers can generate several builds in a day. Also, since tests are only introduced when necessary, it is seen as a Just-In-Time (JIT) technique. When responding to changes, tests have to be changed, and the is code refactored, which helps to reduce technical debt [35].

We now consider how these advantages can be maintained, in a development that uses Event-B. We have highlighted that ProR can be used to link requirements to models, so that aids traceability. There are also tools available for automatic code generation [36], [29], and model-based testing [37], [38], that can be used to generate tests. This would make it amenable to continuous integration. Event-B can also be considered as a JIT approach since abstraction and underspecification are ideal ways of deferring the addition of specific details. Refactoring is also possible with Event-B, and improving the tool support for this is ongoing. Increased confidence in the correctness of code could be gained by generation of verification conditions, such as JML [39] for Java.

IV. Using Event-B in a DAD Project

We consider that DAD and Event-B may be of interest to organizations developing safety-critical systems. We discuss how Event-B might be used in an agile way, as part of a DAD-driven project. In DAD, emphasis is placed on the delivery of consumable solutions, which are a fully working revision of the system. However, we view models, diagrams, simulations, and animations, as consumable solutions; these are delivered at the end of each iteration. Therefore, we consider that working code is not the only measure of progress. This is different from the standard DAD approach where only a fully working revision of the system is considered to be a deliverable. It is often the case, that artefacts such as proven models, test results, and simulation results, are necessary to build the safety case that is required for certification of a critical system [40]. We believe that Event-B and DAD complement each other well, and we now present our view of why agile methods, and DAD, in particular, do so.

We expect it to be beneficial to start using Event-B early in the inception phase of development. Since the DAD life-cycle is flexible, it should be straightforward to incorporate Event-B into the process. However, a development iteration for a formal development may be longer than that of an informal development. This may then require the management of a two-speed development. If this is to be integrated into a process with a non-formal part, care will need to be taken to synchronise the iterations.

There are a number of ways to view, and animate Event-B models, and perform simulations. These should be useful in the inception phase onwards. Using these, teams can be kept informed of the latest developments, which is very important in a team-centric approach. In Figure 5 (which is inspired by the DAD life-cycle diagram (Figure 2) of [41]) we interpret Event-B artefacts as being consumable solutions

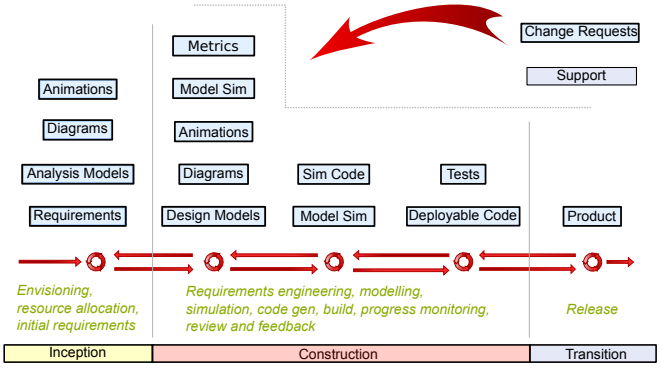

Figure 5. Artefacts Available During an Agile-Event-B Project

of the DAD process. We take the view that stakeholders will be consumers of the artefacts, and the consumables may vary as the project proceeds. When not contributing directly towards product deliverables, they should contribute to making progress throughout the development life-cycle. This includes supporting the process-decision making activity or building a safety-case for certification.

A. Event-B in the Inception and Construction Phases

During the inception stage, we could begin recording requirements using ProR [17]. Requirements can be linked to corresponding Event-B elements, which facilitates traceability. DAD guidelines recommend separating functional and non-functional requirements into different artefacts, in order to focus on the individual needs of each set of requirements. This would be possible in ProR, by generating two requirements specifications. Event-B is most useful for models involving functional, and safety, requirements; whereas non-functional requirements are more suited to simulation and testing. In the latter case, Event-B can assist with multi-simulation [42], [43].

The DAD approach advocates minimal requirements specification but recognises that a fuller treatment is needed for a critical system development, which is likely for projects using Event-B. In Figure 5, we show when various artefacts become available during the various phases. In the inception phase, the system's most important, high-level properties can be stated, in abstract models. These ideas can be communicated to non-technical stakeholders, using animation and other graphical techniques. B-Motion Studio [25] can be used to link images representing domain elements, to the Event-B model. An image's appearance and location can be linked to the state of a model. So, as the model's underlying state changes, then so does the graphical representation. For those requiring more technical details, the ProB animator and model-checker can be used [22]. It provides a simulation feature, where more precise details of the model can be examined. A choice of enabled events is presented to the user, allowing step-by-step analysis of the evolving system. The current state, state history, and event history, are presented. Details of the guard predicates are available, to help resolve event enablement issues. The frequent use of ProB is recommended during modelling and proving.

As the development progresses through the construction phase, state-diagrams [19] and class-diagrams [20] can be used

 ISBN: 978-1-61208-458-9

for specification, and can be used to communicate aspects of the design to other stakeholders. Throughout the development, we can assess more of the non-functional requirements, as the design matures, by simulating Event-B models with more accurate representations of continuous state. Such an approach was developed in the Advance project [42], which focussed on cyber-physical systems, using the Functional Mock-up Interface (FMI) approach [43]. In later stages, code can be generated from the Event-B models, and simulation models can be replaced with Functional Mock-up Units (FMUs) containing code. Then the simulation more closely represents what happens in the actual target platform. Deployable code [36], [29] can be generated automatically from the Event-B models, if desired.

Configuration for specific target platforms can be done during the transition phase, in this case, pre-configured templates, and code injection may help [44]. During the support phase, change requests can be generated. These will be fed back into a new development cycle, as seen in Figure 5.

B. Some Observations on Construction Patterns

During the DAD construction phase, there are a number of patterns (strategies) that could be used. We investigate how they relate to the use of Event-B, exploring a number of them individually.

"The team reliably demonstrates increments of software at the end of each iteration" :- This is the demonstration that an iteration has been successful, but how do we measure success when using Event-B? If code is being generated from the Event-B models, and this code passes all the unit and integration tests, then we could use the same measure of success (by demonstrating the increment). However, if we are not generating code, how is success to be measured?

In order to understand how this strategy would relate to a project using Event-B, it is important to understand why it is considered to be so important in DAD. On closer inspection we find that the important issues are: highlighting problems at an early stage, avoiding scope creep, and resisting perfectionism. The use of Event-B is considered to be a major advantage in helping with the first issue: identifying problems early. The second, limiting scope creep, seems to be independent of the use of Event-B. The last point may be more of an issue with formal development, due to the emphasis on discharging proof obligations. Guidelines could be drawn up about when it is practical to proceed without completing a proof, how to measure the confidence in its correctness, and recording and reporting a proof's status.

We consider the relationship between an incomplete proof and a failing test. Both relate to the failure to satisfy a requirement. On page 297 of [5] it is stated that *"All tests should be implemented before the end of the iteration, for the work to be considered to be potentially consumable"*. However, on page 262, a concession is made in the case-study. At a point in the project, at the end of an iteration, there are still some failing tests. Now, the end-of-iteration date is immutable, so it is accepted that some tests fail. So it would appear that passing all tests should be the goal, but it may not be achievable in the time allowed. A pragmatic approach is to have a two-speed process involving minor-, and major-iterations. All proofs should be completed for major-iterations (which may only happen a few times a year). Due to the weaker requirement associated with minor-iterations, the model, together with adequate documentation, and a measure of model integrity should be sufficient.

"Iteration dates are fixed" :- This point initially seems to be technology-neutral. However, in order to meet this goal, the authors state that it might be necessary to withdraw items from the work-item list. It is suggested that this may work best for larger projects. In small projects, or the small part of a large project that uses Event-B, removal of a work-item from the list may be a more significant part of the iteration deliverable, this could be more of a problem. However, it could be mitigated by flexible working patterns, as recommended in the pattern below.

"Team members who finish their tasks begin another task from the iteration work-list, or help others with their tasks" :- This encourages developers to make the best use of the time available. It would seem to be common sense, but for it to be useful in Event-B, it requires some notion of collaborative working (in Event-B). It would certainly be possible to perform some kind of pair modelling. However, for modellers to be working on independent, parallel tracks, existing composition techniques need to be improved, and this work is being undertaken as part of the ADVICeS project [1]. It includes construction of a useful notion of components, libraries, and interfaces. If such techniques, for parallel development, are not available for Event-B or are ineffective, then Event-B could not really be considered agile with respect to team-working.

"Stakeholders may request a demo (of working software) at any time" :- In a properly versioned Event-B project, using automatic code-generation tools and continuous-integration, this is theoretically possible. It should always be possible to demonstrate *a* build, but it might not be the most recent work. In a project using minor-, and major-iterations, it may be some time between stable builds since minor iterations may be incomplete. From a practical standpoint, formal modelling is not always seen as a process where certified, automatic code-generation is important. If there is no certified, automatic code-generation, then we can still satisfy this goal. But there is less assurance that the high integrity of a formal model has been transferred to the implementation. In addition, if we are using ProB animation, or executable models, as a form of validation, then there is, again, less assurance that what has been validated is transferred to the implementation.

C. Construction Anti-patterns

A number of anti-patterns are described by the authors of DAD. We consider those of interest to us, and describe them as patterns. **"Mitigate risk"**: We assume that the use of Event-B was a step towards achieving this. The authors' advice is, to make sure any concerns are flagged and are prominent in the working environment. **"Prove the architecture, with code, early"**: Would this be too complex a task for Event-B to model and generate code, so early in construction? Event-B may be useful for modelling the architecture early in the project, but this aspect requires investigation. **"Handle missing requirements"**: The discovery of a missing requirement can have repercussions due to dependencies on other requirements. The strategy, in a software-based project, is to provide a stub for the missing functionality or swap out the blocked requirement. In Event-B, requirements are related to model invariants, so,

the missing requirement and its related invariant could simply be added. However, some analysis of the requirement should be done before adding it to the model, to assess validity. We envisage that a process, using Event-B in an agile way, would be able to accommodate this.

V. Process Goals and Event-B

Once the decision to use Event-B has been made, one may wish to examine what development strategies are available, or one may need more detailed guidance on some technical aspects. We have been inspired by DAD's goal-oriented decision approach, to propose a goal-driven process for Event-B development. This could be useful when modelling as part of an agile process, and may be useful to the Event-B community in general.

We consider two categories of potential users: experienced teams and individuals who need a check-list of things to consider, and novices. However, to accommodate continuous improvement in the process, even experienced teams need to be able to record what they have learned, in the form of guidelines, patterns, and components. They then need a way of retrieving that information, so that they may be used to make process decisions, or design decisions, or for use in modelling. Inability to find useful knowledge at an early stage of a project is a barrier to the successful introduction of Event-B. In the case of reusable components, the deployment of components is being investigated [1]. The second category of users is novice teams or individuals. In either case, introducing Event-B into a company's existing development process can be difficult since Event-B advocates can struggle to get accepted into existing company culture.

There has been a great deal of knowledge recorded already, about how to proceed at various stages of the development process. However, this knowledge is widely dispersed on the web, much of it in an unstructured manner. We would like to reduce the number of entry points for obtaining guidance, for the use of Event-B, backed by a goal-driven process and a structured repository of data. However, we are aware that many companies may wish to keep their own private repository too since the information contained within it would be their intellectual property. We should, therefore, attempt to accommodate this in our approach. Another barrier that exists is that much of the knowledge resides behind an academic pay-wall which prevents industry from gaining free access to much of the available information. Our main motivation is to facilitate the creation of a goal-oriented decision-making process. The aim is to guide users, both experienced and inexperienced, providing them with the correct information to make informed decisions. Users need information upon which to base the decisions, and they may also need guidance/reminders about what issues they need to consider.

A. Monitoring mechanisms and metrics

Feedback to developers about the progress of a project is given through monitoring mechanisms, e.g., with the use of metrics and measurements (see Figure 4). They should be easily accessible throughout the duration of the project so that they can provide up to date information, and enable prompt action on any issues that are observed. Collecting product and process measurements early in the development stages is beneficial since it enables the analysis of the quality of the model, as well as improvement of modelling and development process. Furthermore, it contributes to empirically validate the developers perception about the model, and allow better estimation of a project. In [3] we proposed the placement of monitoring mechanisms in short and long iterations of Scrum process. Since the DAD framework extends Scrum, we can use the same metrics and measures, and collect them at the suggested times.

Collecting useful and meaningful measurements, for formal developments used within an agile development process, requires much more than just reusing the metrics commonly used in the programming setting. Not only does one need to consider metrics in terms of requirements and models, but also keep in mind the specifics of formal modelling and agile processes. There are several ground rules regarding the monitoring process: keep it simple and continuous; collect and combine several metrics and look for trends; and, use metrics as indicators rather than facts [5].

Some metrics for agile, and in particular, DAD developments are mentioned in [5]. Below we present the ones that are advocated for DAD and can be easily adapted for Event-B development:

- *Burn rate* depicts how resources are invested in a particular effort, measured by counting, e.g., money or time involved in such an effort. In the case of Event-B modelling, one can use work points assigned to an item (being an estimate of the man-hours necessary to model a requirement or prove a property) or according to complexity estimation (points assigned on the experience-basis). It can be displayed by, e.g., using a burnup chart, where the burnup chart would be comprised of modelled and proven items, and the total amount of work to do on these.
- *Delivered functionality* is the only true measure of progress of a software development project, it is measured by tracking items in a project. In Event-B development observing the modelled and proven items with respect to the ones left in the project backlog would give the full picture on the state of the project. Progress of modelling in Event-B can also be measured with statemachine-based metrics [45].
- *Velocity* describes how much functionality the team can deliver per iteration. It is team-specific and measured in the form of counting "use case points" or "story points". For our purpose, counting requirements modelled and proved, or work points per iteration would show the velocity of Event-B development.

Note that the term *item* stands for a requirement, and/or a property.

As for quality aspect of a development, it is proposed to measure the number and severity of defects. However, in the case of Event-B developments, this metric is not meaningful, as the measurement should be kept on the model level. We propose to use several product measurements, e.g., for the size and complexity of a model, based on Event-B syntax [46] for each refinement step. Moreover, the number of proof obligations, both automatic and interactive, will indicate the complexity of modelling (and proving). Furthermore, adherence to modelling conventions and styles can be collected via model inspections and displayed in a form of a histogram or line chart.

 ISBN: 978-1-61208-458-9

Very much advocated for agile developments, *lifecycle traceability* (to requirements, design, etc.) is enabled by following the refinement of the artefacts of interest. This measurement also shows that product quality is heavily impacted by the quality of the process.

There are several process-related, time-based, metrics that can be useful for Event-B developments. These are, for instance, *activity time* – the time it takes to model and prove an item (requirement or property); *time invested* – the amount of time spent on a project, where work can be categorised by activities like modelling, proving, re-modelling; and *change cycle time* – the period of time from when a requirement, a property or enhancement request is identified, until it is resolved (cancelled, or modelled and proved). All of these metrics can be displayed as a trend on a histogram or line chart.

B. Other Guidance for Modellers/Developers

There are several other approaches that could contribute towards the decision-making process, e.g., in preparation for the iteration planning activity of Figure 4. Kobayashi et al. propose the use of a method to guide refinement strategies [47]. The dependencies between domain elements (called phenomena) and artefacts of the model (which are Event-B invariants) are analysed. Domain elements are modelled, as usual in Event-B, using sets, variables constants, and events. The analysis involves computing the relationships between the domain elements in the invariants, and comparing the order in which domain elements can be added to the model, with the aim of simplifying the refinement. From the analysis, a weighted list of refinement plans is derived, which can help guide development. In further work [48], the authors propose a systematic approach to discount invalid refinement plan proposals.

Another approach is to use existing refinement patterns to propose alternative refinement plans when proof fails [49]. In this approach, a reasoned modelling critic is used to analyse the model, and failing proofs. Under the covers, a search is performed to find the closest match(es) of the current model to existing patterns. Alternative modelling strategies, in the form of high-level refinement plans, are suggested to solve the problem. This approach provides feedback to the user on a day-to-day basis; and can also provide feedback in the high-level, strategic, decision process.

VI. Related Work

Agile methods have established themselves in industry as a diverse range of practices, aimed at improving the software development process. The use of agile methods in critical-systems developments, in particular, with formal methods, has been investigated. For instance, the work of Eleftherakis and Cowling describes XFun [50], an agile approach to formal development using the X-machine formal method. Paige and Brooke [51] show how an integrated formal method (Eiffel [52] and Communicating Sequential Processes (CSP) [53]) can be developed in an agile manner. However, here the emphasis is on methods engineering. More generally, Gary et al. [54] discuss their experience of using agile methods in a critical-system development. They conclude that there is scope for more research in this area. The Safety Critical Systems Club discuss the practicalities of using agile development in the real world [55], with contributions from various developers from the safety-critical systems business. They conclude that some agile practices are already in use in a number of the companies involved, but they are used with a great deal of caution. Wolff describes how the Vienna Development Method (VDM) could be used in a project that uses Scrum [56]. Here, the formal approach is used in parallel with traditional software engineering methods. The formal approach can be used to communicate details about the critical parts of the system to the implementers. In other work, Black et al. posit a positive, but cautious, outlook for an agile/formal mix, in [57], as do Larson et al. in [58].

VII. Conclusion: what is stopping Event-B from being used in an agile development?

We believe that, in theory, Event-B is suitable for use in an agile project. We know that agile methods are being used (at least to some extent) in safety-critical systems development [55], so we would expect that Event-B might be useful here, too. In the technical report [2], a preliminary assessment was performed, to determine which aspects of Event-B could be considered to contribute to an agile approach. A preliminary case-study demonstrating the use of Event-B with agile methods is described in [59]. There are a number of improvements that could be made to make Event-B more agile. Some are tooling issues, like improving refactoring, a major feature of the agile approach. Other improvements require theoretical advancement, as well as tool support, such as the development of reusable components or new diagrammatic representations. In addition to improvements to theory and tools, we may also seek to improve the modelling process. When considering how Event-B could be used as part of an agile engineering process, we believe that a process-decision framework, similar to the one advocated by DAD, would be useful. The aim would be to provide developers (both beginners and experts) with structured assistance throughout the development life-cycle.

When considering how DAD and Event-B complement each other or conflict, we ask many questions. For instance, how do we interpret *consumable solutions* when developing critical-systems with a formal method? Do they have to involve working code, as required in a mainstream software development with DAD? This is particularly relevant for embedded systems, where the ultimate deliverable involves hardware. In this case, simulation is used to postpone commitment to hardware, until the latest possible stage. So we take the view that consumables are artefacts that contribute (in a timely manner) to the ultimate success of a project. They may be of use to different stakeholder audiences; for instance, ProB animation for technically-minded developers, B-Motion Studio for the non-technical, metrics for project planners, and so on. In critical-system development, it may be acceptable to defer code generation, and use animation to perform on-going validation, since it is so important that everybody fully understands the issues involved. Code can be generated automatically from Event-B, but the refinement process, down to the implementation-level, may take longer; and the generated code might provide a less clear result, than model animation. It should be easier to make changes to an abstract model too, since unnecessary detail is omitted from the model, making this more agile.

 ISBN: 978-1-61208-458-9

We also need to understand how formal modelling culture would work in an agile world. An example of this difference might be that between failed tests, and failed proofs. At the end of an iteration, in an agile development, it can be acceptable to have some tests that fail (with agreement from the client). It is understood that as the development proceeds more is learned, which will help resolve the issue, downstream. It would seem, therefore, that it should be acceptable too, to have failed proofs at the end of an iteration; but this sits much less comfortably in the formal world, where so much effort is invested in discharging proof obligations. It may be considered that it is enough to be reasonably confident that a proof will succeed, even though the proof remains to be completed. If this is the case, how do we estimate and record our confidence, and how do we measure progress in light of this? In safety-critical systems development, one would almost certainly need the minor- and major-iteration approach if it is decided that all proofs must succeed for the safety-case to be acceptable; without feedback from the minor-iterations, the development process would not be very agile.

Team-working is a fundamental aspect of an agile project. In an Event-B project, this means that models must be worked on concurrently. We hope to do this with component-based development, which also promotes reuse. Composite components are components made with machines and other components. It should be possible to independently refine at the component-level. However, independent refinement, at a finer granularity (i.e, the sub-models of a component) may be problematic, due to the dependencies introduced by properties specified over the state of the whole component. We are currently exploring the issues, and we are developing a new notion of interfaces for Event-B in the ADVICeS project [1].

The work to maintain and improve Event-B is ongoing. We have identified that we may make a contribution by designing a goal-driven process decision framework, backed by a structured repository for guidelines, and practical advice about development strategies. This would supplement the existing support approach, which is in the form of a wiki. This idea is inspired by the DAD notion of a goal-driven process. The framework could target beginners and experts separately. It should provide help and guidance for novices; for expert users, the approach could provide a checklist of things to consider. The repository could include information about development patterns, guidelines for development strategies, and have a component library. Ideally, for use in industry, there should be an option to set up a private repository too, which might be required to protect a company's IP.

VIII. ACKNOWLEDGEMENTS

This work was carried out within the project ADVICeS, funded by Academy of Finland, grant No. 266373.

REFERENCES

[1] The ADVICeS Team, "The ADVICeS Project," available at https://research.it.abo.fi/ADVICeS/.

[2] M. Olszewska and M. Walden, "FormAgi - A Concept for More Flexible Formal Developments," Tech. Rep. 1124, 2014.

[3] M. Olszewska and M. Walden., "DevOps Meets Formal Modelling in High-Criticality Complex Systems." in In Proceedings of QUDOS2015 workshop within European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), 2015.

[4] J. Abrial, Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.

[5] S. Ambler and M. Lines, Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press, 2012.

[6] R. Jeffries, A. Anderson, and C. Hendrickson, Extreme Programming Installed. Addison-Wesley Professional, 2001.

[7] K. Schwaber and J. Sutherland, "The Scrum Guide, from scrum.org," 2010.

[8] M. Poppendieck, "Lean Software Development," in Companion to the proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society, 2007, pp. 165–166.

[9] J.R. Abrial et al., "Rodin: An Open Toolset for Modelling and Reasoning in Event-B," Software Tools for Technology Transfer, vol. 12, no. 6, Nov. 2010, pp. 447–466. [Online]. Available: http://dx.doi.org/10.1007/s10009-010-0145-y

[10] "The Rodin User's Handbook," Available at http:// handbook.event-b.org/.

[11] A. Romanovsky and M. Thomas, Industrial Deployment of System Engineering Methods. Springer, 2013.

[12] S. Hallerstede, "Justifications for the Event-B Modelling Notation," in B, ser. Lecture Notes in Computer Science, J. Julliand and O. Kouchnarenko, Eds., vol. 4355. Springer, 2007, pp. 49–63.

[13] R. Silva and M. Butler, "Shared Event Composition/Decomposition in Event-B," in FMCO Formal Methods for Components and Objects, November 2010. [Online]. Available: http://eprints.soton.ac.uk/272178/

[14] M. Butler and I. Maamria, "Practical Theory Extension in Event-B," in Theories of Programming and Formal Methods, ser. Lecture Notes in Computer Science, Z. Liu, J. Woodcock, and H. Zhu, Eds. Springer Berlin Heidelberg, 2013, vol. 8051, pp. 67–81. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39698-4_5

[15] J. des Rivieres and W. Beaton, "Eclipse Platform Technical Overview," Available at http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html.

[16] "The Rodin Plug-ins Wiki," at http://wiki.event-b.org/index.php/Rodin_Plug-ins.

[17] M. Jastram, "ProR, an Open Source Platform for Requirements Engineering Based on RIF," available at http://deploy-eprints.ecs.soton.ac.uk/245/1/seisconf.pdf, 2010.

[18] R. Murali, A. Ireland, and G. Grov, "A Rigorous Approach to Combining Use Case Modelling and Accident Scenarios," in NASA Formal Methods, ser. Lecture Notes in Computer Science, K. Havelund, G. Holzmann, and R. Joshi, Eds. Springer International Publishing, 2015, vol. 9058, pp. 263–278. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17524-9_19

[19] V. Savicks and C. Snook, "A Framework for Diagrammatic Modelling Extensions in Rodin," in Rodin Workshop Proceedings, 2012, pp. 31–32.

[20] C. Snook and M. Butler, "UML-B and Event-B: An Integration of Languages and Tools," in The IASTED International Conference on Software Engineering - SE2008, February 2008. [Online]. Available: http://eprints.ecs.soton.ac.uk/14926/

[21] M. Plaska, M. Walden, and C. Snook, "Documenting the Progress of the System Development," in Methods, Models and Tools for Fault Tolerance. Springer, 2009, pp. 251–274.

[22] M. Leuschel and M. Butler, "ProB: An Automated Analysis Toolset for the B Method." STTT, vol. 10, no. 2, 2008, pp. 185–203. [Online]. Available: http://dblp.uni-trier.de/db/journals/sttt/sttt10.html#LeuschelB08

[23] D. Déharbe, "Integration of SMT-solvers in B and Event-B Development Environments," Science of Computer Programming, vol. 78, no. 3, 2013, pp. 310–326.

[24] C. Métayer, "AnimB Homepage," http://www.animb.org/index.xml.

[25] L. Ladenberger, J. Bendisposto, and M. Leuschel, "Visualising Event-B models with B-motion Studio," in Formal Methods for Industrial Critical Systems. Springer, 2009, pp. 202–204.

[26] A. Iliasov et al., "Supporting Reuse in Event B Development: Modularisation Approach," in Abstract State Machines, Alloy, B and Z. Springer, 2010, pp. 174–188.

 ISBN: 978-1-61208-458-9

[27] V. Savicks, M. Butler, J. Bendisposto, and J. Colley, “Co-simulation of Event-B and Continuous Models in Rodin,” in 4th Rodin User and Developer Workshop, June 2013. [Online]. Available: http://eprints.soton.ac.uk/360400/

[28] J. Bendisposto et al., “ADVANCE Deliverable D4.2 (Issue 2) Methods and Tools for Simulation and Testing I,” The ADVANCE Project, Tech. Rep., 2013.

[29] A. Edmunds and M. Butler, “Tasking Event-B: An Extension to Event-B for Generating Concurrent Code,” in PLACES 2011, February 2011. [Online]. Available: http://eprints.ecs.soton.ac.uk/22006/

[30] V. Rivera and N. Cataño, “Translating Event-B to JML-Specified Java Programs,” in 29th ACM Symposium on Applied Computing, Software Verification and Testing track (SAC-SVT), Gyeongju, Korea, March 24-28 2014, pp. 1264–1271.

[31] I. Dinca, F. Ipate, L. Mierla, and A. Stefanescu, “Learn and Test for Event-B - A Rodin Plugin,” in ABZ, ser. Lecture Notes in Computer Science, J. Derrick, J. A. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, Eds., vol. 7316. Springer, 2012, pp. 361–364.

[32] M. Fowler and J. Highsmith, “The Agile Manifesto,” Software Development, vol. 9, no. 8, 2001, pp. 28–35.

[33] T. Dingsoyr, S. Nerur, V. Balijepally, and N. Moe, “A Decade of Agile Methodologies: Towards Explaining Agile Software Development ,” Journal of Systems and Software, vol. 85, no. 6, 2012, pp. 1213–1221.

[34] S. Ambler and M. Lines, “Going Beyond Scrum: Disciplined Agile Delivery,” Disciplined Agile Consortium. White Paper Series, 2013.

[35] V. Krishna and A. Basu, “Software Engineering Practices for Minimizing Technical Debt,” in Proceedings of the International Conference on Software Engineering Research and Practice (SERP). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013, p. 1.

[36] F. Degerlund, R. Gronblom, and K. Sere, “Code Generation and Scheduling of Event-B Models,” Turku Centre for Computer Science, Tech. Rep. 1027, 2011.

[37] Q. Malik, J. Lilius, and L. Laibinis, “Scenario-based Test Case Generation using Event-B Models,” in Advances in System Testing and Validation Lifecycle, 2009. VALID’09. First International Conference on. IEEE, 2009, pp. 31–37.

[38] A. Stefanescu, F. Ipate, R. Lefticaru, and C. Tudose, “Towards Search-based Testing for Event-B Models,” in Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on. IEEE, 2011, pp. 194–197.

[39] N. Cataño, T. Wahls, C. Rueda, V. Rivera, and D. Yu, “Translating B Machines to JML Specifications,” in Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012, pp. 1271–1277.

[40] P. Bishop and R. Bloomfield, “A Methodology for Safety Case Development,” in Industrial Perspectives of Safety-Critical Systems. Springer, 1998, pp. 194–203.

[41] S. Ambler and M. Lines, “Scaling Agile Software Development: Disciplined Agility at Scale,” available at http://disciplinedagileconsortium.org/Resources/Documents/ScalingAgileSoftwareDevelopment.pdf, May 2014.

[42] The Advance Project Team, “Advanced Design and Verification Environment for Cyber-physical System Engineering,” Available at http://www.advance-ict.eu.

[43] The Modelica Association Project, “The Functional Mock-up Interface,” Available at https://www.fmi-standard.org/.

[44] A. Edmunds, “Templates for Event-B Code Generation,” in 4th International ABZ 2014 Conference, 2014. [Online]. Available: http://eprints.soton.ac.uk/364265/

[45] M. Olszewska and M. Walden, “Measuring the Progress of a System Development,” Dependability and Computer Engineering: Concepts for Software-Intensive Systems: Concepts for Software-Intensive Systems, 2011, p. 417.

[46] M. Olszewska and K. Sere, “Specification Metrics for Event-B Developments,” in Proceedings of the CONQUEST 2010: ”Software Quality Improvement”, I. Schieferdecker, R. Seidl, and S. Goericke, Eds. International Software Quality Institute, 2010, p. 112.

[47] T. Kobayashi and S. Honiden, “Towards Refinement Strategy Planning for Event-B,” arXiv preprint arXiv:1210.7036, 2012.

[48] T. Kobayashi, F. Ishikawa, and S. Honiden, “Understanding and Planning Event-B Refinement through Primitive Rationales,” in Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings, 2014, pp. 277–283.

[49] G. Grov, A. Ireland, and M. Llano, “Refinement Plans for Informed Formal Design,” in Abstract State Machines, Alloy, B, VDM, and Z. Springer, 2012, pp. 208–222.

[50] G. Eleftherakis and A. Cowling, “An Agile Formal Development Methodology,” in Proc. 1st South-East European Workshop on Formal Methods, 2003, pp. 36–47.

[51] R. Paige and P. Brooke, “Agile Formal Method Engineering,” in Integrated Formal Methods. Springer, 2005, pp. 109–128.

[52] B. Meyer, “Design by Contract: The Eiffel Method,” in TOOLS (26). IEEE Computer Society, 1998, p. 446.

[53] C. Hoare, Communicating Sequential Processes. Prentice Hall, 1985.

[54] K. Gary et al., “Agile Methods for Open Source Safety-critical Software,” Software: Practice and Experience, vol. 41, no. 9, 2011, pp. 945–962. [Online]. Available: http://dx.doi.org/10.1002/spe.1075

[55] The Safety Critical Systems Club, “Agile Development for Safety Systems,” https://scsc.org.uk/e346.

[56] S. Wolff, “Scrum Goes Formal: Agile Methods for Safety-critical Systems,” in Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches. IEEE Press, 2012, pp. 23–29.

[57] S. Black, P. Boca, J. Bowen, J. Gorman, and M. Hinchey, “Formal versus Agile: Survival of the Fittest,” Computer, vol. 42, no. 9, 2009, pp. 37–45.

[58] P. Larsen, J. Fitzgerald, and S. Wolff, “Are Formal Methods Ready for Agility? A Reality Check.” in FM+ AM, 2010, pp. 13–25.

[59] M. Olszewska, S. Ostroumov, and M. Walden, “Synergising Event-B and Scrum - Experimentation on a Formal Development in an Agile Setting,” Abo Akademi University, Tech. Rep. 1152, 2016.

 ISBN: 978-1-61208-458-9