Early Experience with Agile Methodology in a Model-Driven Approach

Vinay Kulkarni, Souvik Barat, and Uday Ramteerthkar

Tata Consultancy Services, 54-B Indusutrial Estate, Hadapsar, Pune, India
{vinay.vkulkarni,souvik.barat,uday.r}@tcs.com

Abstract. We are in the business of delivering software intensive business systems using model-driven techniques. Developing suitable code generators is an important step in model-based development of purpose-specific business applications. Hence, it becomes critical to ensure that code generator development doesn't become a bottleneck for the project delivery. After establishing a sophisticated technology infrastructure to facilitate quick and easy adaptation of model-based code generators, we experimented with agile methodology. In this paper, we discuss why pure agile methodology does not work for model-driven software development. We propose a modification to the agile methodology in the form of meta-sprints as a golden mean between agile method and traditional plan-driven method. Early experience with the proposed development method is shared along with the lessons learnt.

Keywords: model-driven development, agile method, software intensive business systems.

1 Introduction

We are involved in developing business-critical software systems for large enterprises. These systems are characterized by low algorithmic complexity, database intensive operation, large size, and distributed architecture. The large size of a typical business application leads to large development team that needs to work in a coordinated manner. Choice of distributed architecture paradigm necessitates effective management of multiple technologies such as databases, online transaction processing monitors, batch schedulers, and graphical user interface platforms. Moreover, many a time the customer has non-negotiable technology platform preferences. To avail the short opportunity window, the solution needs to be delivered quickly, and being business critical in nature, is expected to be in use for a long time. Given the increasing business and technology dynamics, the latter poses a significant architectural challenge. Our experience is that no two solutions, even for the same business intent such as straight-through-processing of trade orders, back-office automation of a bank, and automation of insurance policies administration, are identical. Though there is a significant overlap across functional requirements for a given business intent, the variations are manifold too. Moreover, the higher management expects delivery of subsequent solutions for the same business intent to be significantly faster, better and cheaper.

Use of the model-driven approach helped us to separate functional concerns from technology platform thus enabling developers to focus solely on specification of business functionality in an intuitive manner closer to the problem domain [13]. Use of a component abstraction designed essentially to facilitate divide-and-conquer helped manage the large size. The application could now be modelled as a set of related components with provider – consumer relationship being made explicit. Use of component as a unit of development led to better coordinated development process wherein components could be implemented in parallel with assurance of integration into a well-formed application later [15]. A set of code generators translated component / application specifications into the desired technology platform thus delivering increased productivity, uniformly high code quality, and platform independence [14]. In our experience, no two solutions shared identical choices for design strategies, architecture, and technology platform. Since code generators encode these details while transforming application specs to implementation, every new project necessitated development of new set of code generators. Therefore, MasterCraft team was becoming a bottleneck in fast delivery of the purpose-specific business application. We devised the building block abstraction to specify the desired code generator as a hierarchical composition from which its implementation can be automatically derived [16]. In spite of these advances in the mechanisms for implementing a model-driven approach, we were still somewhat away from the desired agility and responsiveness. This led to us looking into the process aspect of model-driven development.

Agile development method is gaining industry acceptance. We argue the method cannot be used as is with model-driven approaches. We suggest modifications that need to be introduced into the agile method for delivering purpose-specific software systems at product cost. We begin with an overview of our model-driven approach and toolset. We then describe the proposed agile development methodology. We discuss early experience, benefits and lessons learnt before concluding with a summary.

2 Our Model-Driven Development Approach and Toolset

Model-driven development approach starts with defining an abstract specification that is to be transformed into a concrete implementation on a given target architecture [16]. The target architecture is usually layered with each layer representing one view of the system. Typically, business applications are implemented across three layers – user interface, application functionality and database, where a user interacts with an application through its user interface layer, application layer implements the business functionality in terms of business logic, business rules and business process, and database layer provides persistency of an application. The modeling approach constructs the application specification using different abstract views - each defining a set of properties corresponding to the layer it models. The model captures structural concerns and a high level language or meta-model is used to specify behavioral concerns. View specification in terms of the model and the text is transformed into the desired implementation.

We developed a model-driven development toolset, MasterCraft [19], for developing large database-centric business critical applications. It comprises: i) a meta modeling tool to specify an abstract view, ii) a set of modelers to populate an instance of an abstract view, iii) a set of code generators that transform each view instance to the desired implementation artifacts, iv) several build automation utilities, and v) a repository-centric component-based development method. An application specification, specified using the various modelers, can be targeted to multiple technology platforms and different architectures of choice by using suitable code generators and build automation utilities. In essence, model-based code generator interprets the model in the light of suitable design decisions and architectural strategies in order to generate code for a specific implementation technology platform. For instance, a class model can be transformed into database access layer code for JDBC or ODBC or ProC while incorporating suitable O-R mapping and currency management strategies.

Managing evolution of MasterCraft was fairly simple when target platform and technology choices for the generated application were relatively bounded. But of late, with increased number of users the demand for supporting new technologies and evolving architectures has increased significantly. Moreover, we experienced that no two generated applications made the same choices of design decisions, architectural strategies and technology platform. With MasterCraft tools interpreting the choice in order to generate the desired code, addition of a new choice or a new configuration of choices would result in modifications to these tool implementations. Our standard practice was to identify a closest-match version of the tool, create a copy of its implementation, and modify it suitably. One would expect such jump-started approach to be time and effort saving, but our experience was to the contrary – to say nothing of the increased hassles of versioning and configuration management. To overcome these issues, we re-architected MasterCraft such that a code generator is specified as a hierarchical composition of model-to-text transformation templates with well-defined extension points. A plug-in for an extension point is also a model-to-text transformation template which in turn may have its own extension points and so on [2]. This extensible plug-in architecture is implemented on Eclipse [11]. Though this enabled MasterCraft toolset to be maintained as a code generator product line, the complexities related to changes that are not predicted a priori and hence require exploration remained unaddressed. Based on the ease of evolution, we categorize MasterCraft activities into the following three kinds:

- *Extension*: Adding new extension for a predefined extension point. Impact of this kind of activity is typically localized and low. For instance, supporting a new widget such as new grid control in GUI modeler, supporting a different kind of logging capability in generated application.
- *Mutation*: Changes related to the internal structure / architecture / design of a tool so as to add a new extension point and the related refactoring effort. This impact is typically large and knowledge intensive. For instance, re-architecting a tool for the plug-in architecture, externalizing GUI screen flow.
- *Exploration*: Exploratory work for introducing new concepts into MasterCraft and proving them. For instance, code generation for deployment on public cloud.

Typical characteristics of these kinds of activities with respect to overall MasterCraft development effort are described in table 1.

Table 1. Characteristics of MasterCraft Development effort

	Extensions	Mutations	Explorations
Typical demand/year	~100	~10	3-4
Average invested effort with respect to total invested effort	50%	30%	20%
Success rate (Converted into MasterCraft feature in time)	80%	60%	40-50%
Typical turnaround time	6 months (One release cycle)	6 months – 1 years (1-2 release cycle)	> 1 year
Customer focus	High	Low	High-Low

Even with clear understanding of MasterCraft evolution characteristics, use of abstractions for improved change isolation, and use of software product line techniques, the overall turnaround time for delivering new functionality did not improve to the desired degree. This led us to look into the process aspect of MasterCraft evolution. We were using traditional waterfall method [22] along with traditional team structure for managing evolution of MasterCraft toolset. Development activities were characterized by detailed planning and rigorous review process as prescribed by waterfall model. MasterCraft development team comprised of independent teams one each for a tool catering to a specific architectural layer of the generated application, e.g. GUI team, Server side team, DB team. Each team was reasonably small in size and conformed to the traditional organization structure i.e. a team lead, one or two module leads and team members. All team leads reported to a single group leader responsible for the entire toolset. Typical turnaround time was about 6-12 months.

This mode of operation served well in early development of MasterCraft when more than 70% of activities were either exploratory or mutative and the delivery timeframes were more relaxed. With more than 50% of the development activities today being extensions, MasterCraft users naturally demand a far shorter turnaround time for new enhancement requests. We were unable to meet this demand with existing team structure using the waterfall model of development. We discovered several reasons for these limitations. It was difficult to plan for small and semi-volatile requirements. The low value-add activities such as status reporting, tracking meetings and so on made the operational process sluggish. Rapidly changing requirements and lack of coordination between different sub-teams led to high amount of rework. Fewer and far spaced deliveries meant infrequent and delayed user feedback resulting eventually in low team morale. The waterfall model didn't provide the necessary visibility at the desired frequency to customer about the development artifacts. As a result, work reprioritization suffered. In addition, we found that existing methodology left little scope for partial (and incremental) delivery for quick-win (and continuous improvement), and early demonstration of research ideas for end-users' feedback.

3 Proposed Development Methodology

With the principal objectives of delivering functionality that brings value to the customers, establishing better mechanisms for feedback from all stakeholders, and enabling quick transformation of an idea/requirement into a set of MasterCraft features, we found Agile Manifesto [18] as the best bet for many reasons. Existing approach heavily depended on documentation for communication between the phases that, we felt, could be eliminated to some extent by having more and closer interactions of customer with tool and solution builders – it was always a demand to show some working software than say, a usecase diagram or a design document. In existing approach, one could get to see a working version after a significant time has elapsed after the requirements were communicated. Thus, it was hard to establish quick-wins with the existing approach. Moreover, most certainly the requirements would have undergone a change thus necessitating rework. Agile method puts greater stress on close collaboration with the customer as opposed to a contract. As advised in Agile methodology, we felt that responding to a change requisitioned by the customer should take precedence over following a plan.

However, we found some limitations of using Agile methodology for all kinds of MasterCraft development activities. Agile methods haven't been as useful for large development teams comprised of members having wide variance in expertise levels and operating in a geographically distributed manner [10]. In addition, some characteristics of MasterCraft created hindrances for applying Agile methodology uniformly. We observed that *mutation* and *exploration* kind of changes are not suitable for Agile method. Catering to mutative changes demands in-depth analysis, experimentation and detailed documentation of the results, observations and conclusions. These activities are difficult to achieve in the short sprint cycles advocated by Agile method. Exploratory work demands in-depth study and analysis which is hard to synchronize with sprint timelines. Agile method advocates to keep customer aware of all decisions, however, that may create unnecessary pressure during exploratory stage and also entail significant product testing and quality assurance effort at the exploratory stage itself. Agile method puts greater stress on working software as opposed to documentation, however, low / no documentation of design rationale and far reaching changes may create problems for hassle-free maintenance and smooth induction. Customer visibility into sprint-backlogs and planning means there is little opportunity for long term research activities. Moreover, defining sprint-backlogs based purely on customer requirements is not always possible – as MasterCraft is a set of interrelated tools, sometimes the order of scoping a feature in a sprint-backlog depends on internal factors rather than purely customer needs.

To overcome these concerns, we modified the standard Agile methodology as depicted in Fig. 1. We organized the development process at two levels which execute

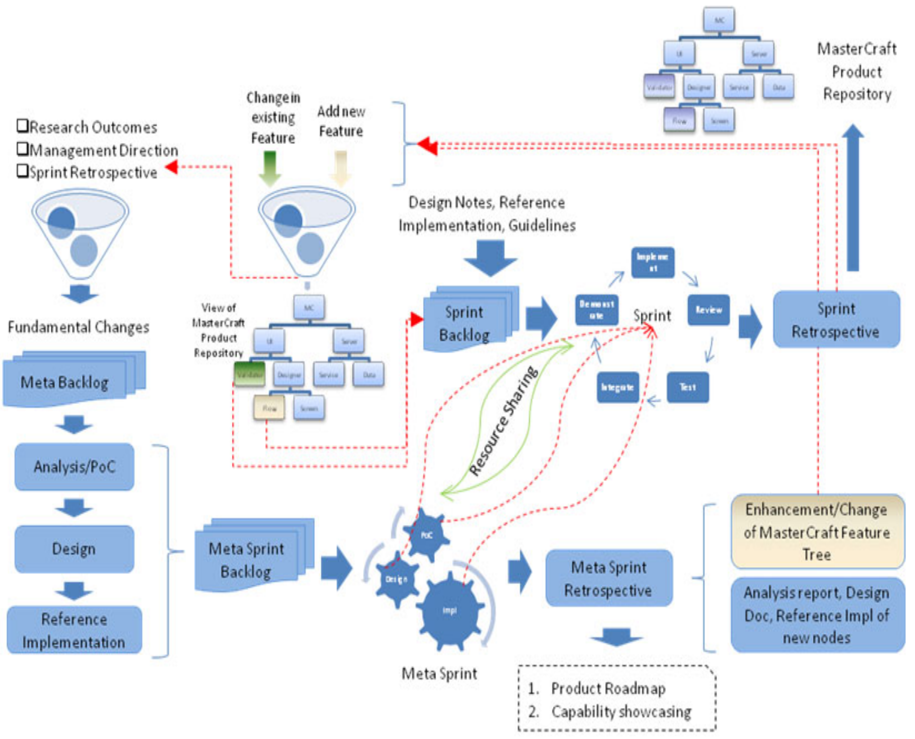

Fig. 1. Adapted Agile Methodology for MasterCraftDevelopment

as parallel threads having periodic synchronization. A feature backlog comprising of new feature requests and change requests drives the development process [7]. Items that are well-understood are prioritized to be taken up for implementation through a series of normal sprints. Changes that are more fundamental in nature and require detailed exploration or even research are prioritized to be taken up for implementation through a series of longer duration sprints that we term as meta-sprints. Meta-sprints differ from normal sprints in that they don't necessarily produce working software as deliverable. Instead, meta-sprints carry out precise investigations the results of which enable product evolution through normal sprints which is similar to SCRUM [24] iteration. Essentially, meta-sprint is to understand the problem statement, to determine the *what* part of the solution, and to break the what part into smaller units of work (work breakdown structures) such that these can be scoped in sprint iteration(s). Meta-sprint also deals with the inherent precedence amongst the units of work. Typically, meta-sprints are of a longer duration than normal sprints. For instance, we are gravitating towards a 12-week meta-sprint whereas normal sprint lasts for 4 weeks. Unlike Agile method, meta-sprint deliverables are not working software but can be a design document, a proof-of-concept implementation, evaluation of a set of design strategies, work breakdown structures, or prioritization preferences for sprints

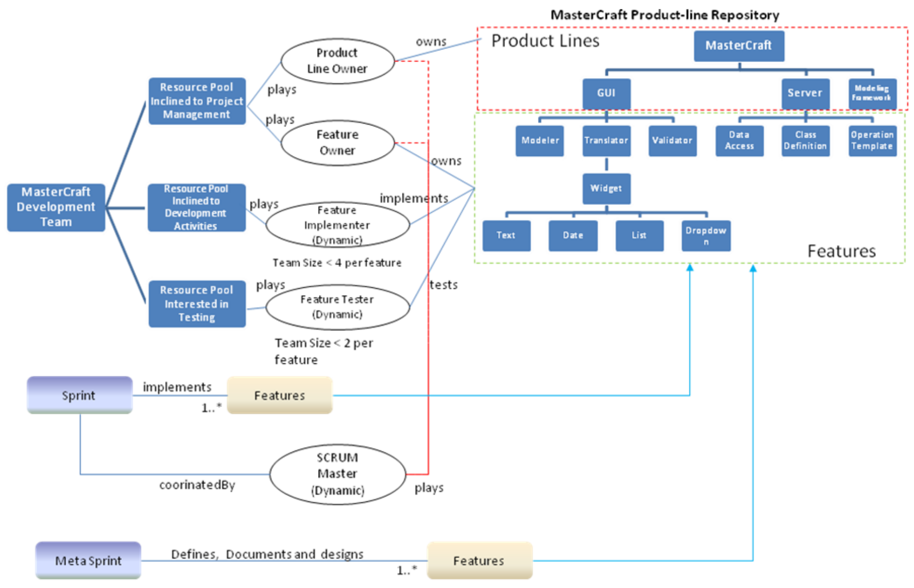

Fig. 2. MasterCraft productline team structure with roles and responsibilities

iteration. Unlike traditional method, meta-sprint puts an upper bound in terms of time (and hence effort) on exploratory and mutative activities. In addition, meta-sprint deliverables facilitate subsequent normal sprints. Thus, development proceeds on two parallel threads namely meta-sprint and sprint with periodic synchronization between the two threads. We use a planning technique, similar to Timeboxing planning technique [12], for synchronizing instances of the two threads. Our planning technique emphasizes on agreed timeline for deliverables instead of the scope of the deliverables i.e. compromise on the scope of the delivery of an iteration to maintain the timeline. Both the threads share a common feature list or sprint-backlog. Theoretically, one can possibly think of meta-meta-sprint, meta-meta-meta-sprint and so on. However, in our experience, so far two levels seem to suffice.

We used feature model notation to declaratively state MasterCraft capabilities. Since MasterCraft code generators themselves are specified declaratively in a model form, establishing traceability from the feature model to code generator specs was relatively straightforward [16 and 17]. Repository-centric model-driven nature of MasterCraft ensured that its feature model can act as the sole driver for the evolution process. Meta sprint delivers working prototypes/concept notes and a proposed feature tree with new feature/option whereas sprint delivers a working MasterCraft with new features/options. We used burn-down charts as an indicator of health and hygiene of overall MasterCraft.

Since developing suitable code generators is an important step in model-based development of purpose-specific business applications, it becomes critical to ensure that code generator development doesn't become a bottleneck for the project delivery. We used meta-sprints primarily to manage evolution of MasterCraft and normal sprints to manage delivery of application using MasterCraft. Therefore, meta-sprints were needed only in the initial stages of the project delivery. However, there were occasions when meta-sprints were resorted to in the light of significant changes requested by customer at a later stage of project delivery. The two level process, and repository-centricity and model-driven nature of both MasterCraft and application development help address such changes in a tractable manner.

Development team was restructured to make everyone accountable and responsible for delivering MasterCraft feature(s). We restructured our development team as shown in Fig. 2. Essentially, we moved the ownership from Project Lead / Module Lead to a feature owner for delivering or exploring a feature, and scrum–master for executing iterations. To improve involvement, we encouraged members to play different roles for different sprints. We encouraged all stakeholders to participate in decision-making. Essentially, sprint flow is a minor adaptation of SCRUM methodology. In addition, we set some rules for smooth execution of sprints. For instance,

- All stakeholders to be involved in scoping the sprint backlog.
- Items exceeding 5% schedule slippage to be automatically dropped from the current sprint-backlog.
- Each sprint to produce an *adequately tested* working version.
- A sprint to last for 4 weeks out of which- 3 weeks to be reserved for development, internal testing and review, and one week for integration and integration testing.
- Stand-up meetings to be conducted as and when required but at least twice a week.

4 Early Results, Benefits and Lessons Learnt

With the proposed development method, we observed slow but steady improvement in delivery of features on time. In first iteration we delivered only 50% of the promised sprint-backlog with a delay of 2 weeks. But results improved significantly in subsequent iterations, and we could achieve our target on time within 3 sprints. We conducted many sprints of duration 4-5 weeks and a few meta-sprints of duration 2-3 months. About 15-20 features were implemented in each sprint and 1-2 research ideas/mutative changes were taken up in each meta-sprint. The usual sprints worked well for *extension* kind of evolution with fairly accurate effort estimation. We tried to use normal sprint for *mutation* and *exploration* kinds of activities, but the result was not very positive. However, those worked well with the proposed approach with improved turn-around as compared to the existing approach. Sprint and meta-sprint bring several tangible and intangible values to the overall development. Those are documented in table 2.

Several literature surveys, case studies and our early experience essentially suggest a limited scope of using Agile methodology, i.e. it is more effective in a context where development infrastructure and development team are matured, development

Table 2. Deliverables and benefits of Sprint and Meta-sprint

	Sprint	Meta-Sprint
Turnaround time from requirement to delivery	4 weeks as compared to 6 months for traditional approach	3 months as compared to minimum 6 months for traditional approach
Productivity	High due to continuous focus on deliverable unit and better issue resolution.	Better as research and exploratory efforts channelize through proper execution path.
Customer Expectation Management	A demonstrable version with latest feature is always available	A demonstrable PoC/Prototype is available
Resource Utilization	High	Not changed from traditional approach.
Team morale	High (nobody, specially juniors, feels left out at any time of development activity)	High due to more frequent interaction with end-users and early feedback.
Rework effort	Low due to early feedback	Low due to early demonstration with a working prototype.

architecture and core design decisions are proven, and requirements are relatively dynamic but less critical in terms of rework for any change. The effective context of Agile methodology with respect to traditional methodology along with the increased scope of the context while adapting meta-sprint flow is depicted in Fig. 3. It worked well for mutative and exploratory activities. However we identified several challenges that need to be addressed for better execution of sprints and meta-sprints,

Team Maturity: Sprint and meta-sprint both work well for teams strong on knowledge and experience. However, inducting new people in the team becomes a challenge as every team member is fully occupied throughout the iteration.
Automated Testing: Lack of automation in integration testing led to longer sprint durations.
Configuration Management: Typically, many parallel teams are working for different sets of features; hence a better configuration management tool is required.
Document Generation: As the method puts more stress on producing working software over documentation, we resorted to generation of minimal documentation from the models. As MasterCraft code generators are also generated from their model specifications, this strategy sufficed for documenting generated applications as well as the code generation toolset.
Migration of Models: Some change requests involved change in the meta models. These changes resulted in a side-effect – earlier models had to be migrated to the new meta model. Here, at times, the short sprint cycles were a challenge.

There were many lessons learnt while moving a large project from traditional development method to the proposed development method. We took time to stabilize into the new method. Initial estimates were off by a large margin and burn charts were always red. Working on smaller chunks with frequent synchronization resulted in on-time and high quality delivery. Non-technical issues like coordination, motivation and

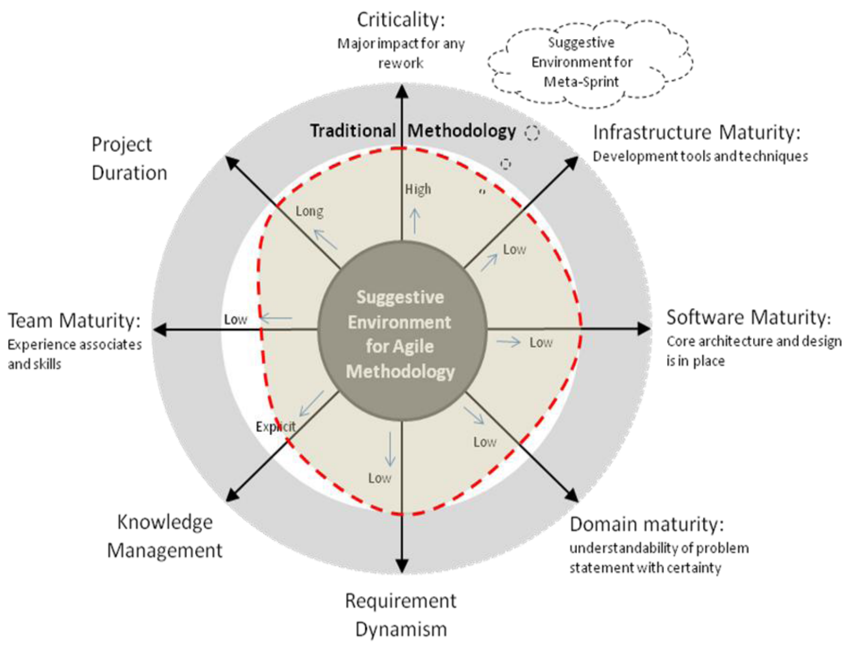
Criticality:
Major impact for any rework
Suggestive Environment for Meta-Sprint
Traditional
Methodology
Infrastructure Maturity:
Development tools and techniques
Project Duration
Long
High
Low
Team Maturity:
Experience associates and skills
Low
Suggestive Environment for Agile Methodology
Low
Software Maturity:
Core architecture and design is in place
Explicit
Low
Low
Knowledge Management
Domain maturity:
understandability of problem statement with certainty
Requirement Dynamism

Fig. 3. Environment of different methodologies and fitment of meta-sprint

taking ownership seemed to be the key success factors. We experienced the necessity of a motivated scrum master in the transition phase for bringing an attitude change in the entire team. We learnt not to be particular about the process – whether to use agile, traditional or something in-between like meta-sprint. Letting the requirement decide the development process rather than any organizational diktat seemed to work. For example MasterCraft uses product line architecture to guide product evolution. Thus visualization of requirements in the form of a feature model is very intuitive. Early experience indicates that a hybrid approach is better for projects in incubation stage. Today, MasterCraft is a mature software system with more than 50% work being of extension kind. Therefore, pure Agile methodology works well. It might not have worked in the early stages where core architecture and design were being formulated.

5 Related Work

Numerous development methods and models have been proposed for developing applications in a systematic and efficient manner. Fundamentally, they are categorized into two kinds – Plan-driven approach and Agile approach. The Plan-driven approach, like waterfall model, spiral model, V-model, focuses on stability and higher assurance for a predefined sets of requirements. On the other hand, Agile approach, such as XP [3], Crystal Clear [8], Dynamic System Development Method [23] Feature Driven Development [21], and SCRUM [24], advocates faster development with increased customer satisfaction for dynamic requirements using iterative and incremental development techniques [4]. As argued in [4] we also experienced that there are some home grounds for pure plan-based approach and Agile approach but appropriate balancing of these two approaches provides better handle for managing dynamic requirements with increased stability and assurance. In the literature, several tailored development approaches are recommended for developing applications such as Lean/Agile development methodology [6], and SCRUM and CMMI based development methodology [26]; and few approaches are also presented to systematize these tailoring processes with increased precision [5, 9]. However these tailored approaches address development of one-off application only. The need of delivering a set of applications that vary along multiple dimensions is not met. We presented a tailored approach to address this need of managed evolution using model-based techniques. The proposed meta-sprint follows agile philosophy rather than any specific process, such as SCRUM, XP, FDD. The core differences of sprint and meta-sprint flow are discussed in table 3.

Table 3. Sprint Vs Meta-Sprint

	Sprint	**Meta-sprint**
Qualifying criteria	No impact on the core architecture/design or concept.	Fundamental changes to MasterCraft
Output	Working MasterCraft with new features	Approach note, reference implementation of features, prospective feature tree
Timeline	4-5 weeks (fixed)	2-3 months (varies)
Communication	Informal	Formal but not in specific format
Knowledge Management	Tacit	Explicit
Visibility	To all stake-holders	Opaque to external stakeholders, e.g. customers.

Essentially, meta-sprint relaxes the criteria of plan-based approach by limiting the planning only for strategic and high-level activities, and monitoring them in terms of observable outcomes instead of recording day-to-day progresses. Similarly, it relaxes some of the mandates of Agile approach such as working software as a deliverable at the end of each cycle, 3-6 weeks cycle time and total visibility to all the stakeholders. With these adaptations in development methodology, we could overcome the limiting factors [10, 20] of both kinds of methodologies while retaining the benefits of both. We adapted Timeboxing technique [12] suitably to synchronize different iterations of meta-sprint and sprint threads. On the other hand, the meta-sprint differs from meta-SCRUM [25] to some extent as meta-sprint details a requirement and analyzes it further to break a larger activity into smaller units that can be accommodated in sprint whereas meta-SCRUM synchronizes and plans several interrelated sprints to manage them flawlessly.

The use of Agile methodology in model-driven development is not very prevalent yet, except tailored Agile approaches, such as Agile model driven development [1]. However these approaches are restricted to modeling activities rather than emphasizing on delivering entire application from the models. On the contrary, we argue that true agility in model-driven development is possible only when code generators can also be adapted as quickly as application models.

6 Summary

We are in the business of delivering software intensive business systems using model-driven techniques. Since developing suitable code generators is an important step in model-based development of purpose-specific business applications, it becomes critical to ensure that code generator development doesn't become a bottleneck for the project delivery. After having put in place sophisticated technology infrastructure in place to facilitate quick and easy adaptation of model-based code generators, we experimented with agile methodology. We discussed why pure Agile methodology does not work for model-driven software development. We proposed modification to the Agile method in the form of meta-sprints as a golden mean between Agile method and traditional plan-driven method. Unlike Agile method, meta-sprint deliverables are not working software but could be a design document, a proof-of-concept implementation, evaluation of a set of design strategies etc. Unlike traditional method, meta-sprint puts an upper bound in terms of time (and hence effort) on exploratory activities. In addition, meta-sprint deliverables facilitate subsequent normal sprints. Thus development proceeds on two parallel threads namely meta-sprint and sprint with periodic synchronization between the two threads.

Early results of using Agile methodology are encouraging with a note that it is not applicable for all kinds of development activities and needs considerable preparedness for deployment in practice. We adapted true agile methodology by introducing meta-sprint concept for mutative and exploratory work; and used this methodology only after plug-in architecture and suitable tools were in place. Though our objective is to channelize more development activity through sprint stream than meta-sprint stream and establish an agile development environment, we would like to continue with a relatively relaxed environment (meta-sprint) for exploratory and knowledge intensive activities. Our early results show that this kind of hybrid development environment is better suited for model-driven software development.

References

1. Ambler, S.W.: Agile Model Driven Development, AMDD (2007), `http://www.agilemodeling.com/essays/amdd.htm`
2. Barat, S., Kulkarni, V.: Developing configurable extensible code generators for model-driven development approach. In: SEKE, pp. 577–582 (2010)
3. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, MA (2000)
4. Boehm, B., Turner, R.: Observations on Balancing Discipline and Agility. In: Proceedings of the Agile Development Conference, ADC 2003. IEEE Computer Society, Los Alamitos (2003)

5. Cao, L., Mohan, K., Xu, P., Ramesh, B.: A framework for adapting agile development methodologies. EJIS 18(4), 332–343 (2009)
6. Cawley, O., Wang, X., Richardson, I.: Lean/Agile Software Development Methodologies in Regulated Environments – State of the Art. In: Abrahamsson, P., Oza, N. (eds.) LESS 2010. LNBIP, vol. 65, pp. 31–36. Springer, Heidelberg (2010)
7. Cockburn, A.: Agile Software Development. Addison-Wesley Professional, Reading (2001) ISBN 0-201-69969-9
8. Cockburn, A.: Crystal Clear, A Human-Powered Methodology for Small Teams. Addison-Wesley Professional, Reading (2004) ISBN 0-201-69947-8
9. Conboy, K., Fitzgerald, B.: Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion. ACM Trans. Softw. Eng. Methodol. 20(1) (2010)
10. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information & Software Technology 50(9-10), 833–859 (2008)
11. Eclipse, `http://www.eclipse.org/`
12. Jalote, P., Palit, A., Kurien, P., Peethamber, V.T.: Timeboxing: a process model for iterative software development. Journal of Systems and Software 70(1-2), 117–127 (2004)
13. Kulkarni, V., Venkatesh, R., Reddy, S.: Generating enterprise applications from models. In: Bruel, J.-M., Bellahsène, Z. (eds.) OOIS 2002. LNCS, vol. 2426, pp. 270–279. Springer, Heidelberg (2002)
14. Kulkarni, V., Reddy, S.: Model-Driven Development of Enterprise Applications. UML Satellite Activities, 118–128 (2004)
15. Kulkarni, V., Reddy, S.: Introducing MDA in a large IT consultancy organization. APSEC, 419–426 (2006)
16. Kulkarni, V., Reddy, S.: An abstraction for reusable MDD components: model-based generation of model-based code generators. In: GPCE, pp. 181–184 (2008)
17. Kulkarni, V.: Use of SPLE to deliver Custom Solutions at Product Cost - Challenges and Way forward. In: Product LinE Approaches in Software Engineering (PLEASE 2011)Conjunction with the 33nd International Conference on Software Engineering, ICSE 2011 (2011)
18. Manifesto for Agile Software Development - agilemanifesto.org/
19. MasterCraft – Component-based Development Environment. Technical Documents. Tata Research Development and Design Centre, `http://www.tata-mastercraft.com`
20. Nerur, S., Mahapatra, R., Mangalaraj, G.: Challenges of Migrating to Agile Methodologies. Communications of the ACM 48(5) (May 2005)
21. Palmer, S.R., Felsing, J.M.: A Practical Guide to Feature-Driven Development. Prentice Hall, Englewood Cliffs (ISBN 0-13-067615-2)
22. Royce, W.: Managing the Development of Large Software Systems. In: Proceedings of IEEE WESCON 26, pp. 1–9 (August), `http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf`
23. Salo, A., Warsta, R.: Agile Software Development Methods: Review and Analysis, vol. 478, pp. 61–68. VTT Publications (2002)
24. Schwaber, K., Beedle, M.: Agile Software Development with SCRUM. Prentice Hall, Englewood Cliffs (2001)
25. Sutherland, J.: Future of scrum: parallel pipelining of sprints in complex projects. In: Agile Conference (2005)
26. Sutherland, J., Jakobsen, C.R., Johnson, K.: Scrum and CMMI Level 5: The Magic Potion for Code Warriors, agile. In: AGILE 2007, pp. 272–278 (2007)