Agile Term Paper

Comparison of 3 Agile Methods of Software Development

Mrunal Phadke
Computer Science Department
Stevens Institute of Technology
Hoboken, New Jersey
mphadke@stevens.edu

Abstract— As the software industry has evolved, so has the need of software development life cycles that are very crucial for delivering the quality software. Starting from the waterfall model (sequential approach) to Agile methods (flexible), many methods have been developed and are used widely by organizations to standardize the processes involved in developing software.

Agile software development focuses on keeping code simple, testing often, and delivering functional bits of the application as soon as they're ready. The goal is to build upon small client-approved parts as the project progresses, as opposed to delivering one large application at the end of the project.

In this paper, we will describe and compare 3 Agile methodologies and eventually decide which one, we think is the best.

1. Scrum

1.1. Description

In the agile Scrum, instead of providing complete, detailed descriptions of how everything is to be done on a project, much of it is left up to the Scrum software development team. This is because the team will know best how to solve the problem they are presented.

1.2. Detailed Process

The following is a stepwise overview of the Scrum Process:

1. Product Backlog - A product owner creates a prioritized wish list called a product backlog.
2. Sprint Planning and Sprint Backlog - During sprint planning, the team pulls a small chunk from the top of that wish list, a sprint backlog, and decides how to implement those pieces.
3. Sprint and Daily Scrum - The team has a certain amount of time — a sprint (usually two to four weeks) — to complete its work, but it meets each day to assess its progress (daily Scrum).
4. Scrum Master - Along the way, the Scrum Master keeps the team focused on its goal.
5. Shippable Product at end of Sprint - At the end of the sprint, the work should be potentially shippable: ready to hand to a customer, put on a store shelf, or show to a stakeholder.
6. Sprint Review and Sprint Retrospective - The sprint ends with a sprint review (analyze the Sprint) and retrospective (make improvements).
7. Next Sprint - As the next sprint begins, the team chooses another chunk of the product backlog and begins working again.

1.3. Management Structure / The Scrum Team

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. The teams are self-organizing and they know how best to accomplish their work, rather than being directed by others outside the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.
Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback.

There are 3 main roles in Scrum:

1. Product owner

   The Product Owner is the sole person responsible for managing the Product Backlog.
   Product Backlog management includes:
   - Clearly expressing Product Backlog items;
   - Ordering the items in the Product Backlog to best achieve goals and missions;
   - Optimizing the value of the work the Development Team performs;
   - Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next; and,
   - Ensuring the Development Team understands items in the Product Backlog to the level needed.

2. Scrum Master

   The Scrum Master is responsible for making sure a Scrum team lives by the values and practices of Scrum.

The Scrum Master is often considered a coach for the team, helping the team do the best work it possibly can.

The Scrum Master does anything possible to help the team perform at their highest level.
His responsibilities involve:

- Removing any impediments to progress;
- Facilitating meetings;
- Coaching the Development Team in organizational environments in which Scrum is not yet fully adopted and understood.
- Doing things like working with the product owner to make sure the product backlog is in good shape and ready for the next sprint.
- Finding techniques for effective Product Backlog management;
- Helping the Scrum Team understand the need for clear and concise Product Backlog items;
- Understanding and practicing agility; and,
- Leading and coaching the organization in its Scrum adoption;
- Planning Scrum implementations within the organization;
- Helping employees and stakeholders understand and enact Scrum and empirical product development;
- Causing change that increases the productivity of the Scrum Team; and,
- Working with other Scrum Masters to increase the effectiveness of the application of Scrum in the organization.

The Scrum Master role is commonly filled by a former project manager or a technical team leader but can be anyone.

3. Development Team

The development team is a group of developers who are responsible for designing and developing the “Done” product at the end of each Sprint.

The team has following characteristics:

- They are self-organizing and they decide how to turn Product Backlog into Increments of potentially releasable functionality;
- Development Teams are cross-functional with all the skills as a team necessary to create a product Increment;
- Scrum recognizes no sub-teams in the Development Team, regardless of specific domains that need to be addressed like testing or business analysis;
- Individual Development Team members may have specialized skills and areas of focus, but accountability belongs to the Development Team.

1.4. Alignment with Agile Manifesto

1. Individuals and interactions over processes and tools

Scrum is a team-based effort to delivering value to the business. Team members work together to achieve a successful deliverable product. The Scrum framework promotes effective interaction between team members so the team delivers value to the business.

Once the team gets a business goal, it designs and develops the plan and works on it collaboratively to deliver the desired product. In this process the team strives to solve all the difficulties together and focuses on getting work done. This focus on team responsibility in Scrum is critical.

2. Working software over comprehensive documentation

At end of every Sprint, Scrum requires a working, finished product increment. A Scrum team’s goal is to produce a product increment every sprint. The increment may not yet include enough functionality for the business to decide to ship it, but the team’s job is to ensure the functionality present is of shippable quality.

3. Customer collaboration over contract negotiation

Scrum is a framework designed to promote and facilitate collaboration. Team members collaborate with each other to find the best way to build and deliver the software, or other deliverables, to the business. The team, especially the product owner, collaborates with stakeholders to inspect and adapt the product vision so the product will be as valuable as possible.

4. Responding to change over following a plan

Scrum teams make frequent plans such as Sprints. In addition, many teams create longer-term plans, such as release plans and product roadmaps. These plans help the team and the business make decisions. However, the team’s goal is not to blindly follow the plan; the goal is to create value and embrace change. The thought process and ideas necessary for planning are more important than the plan itself.

As plan created early may not be perfect, as the team gathers deeper knowledge and new information is discovered, the team updates the product backlog. That means the direction of the product likely shifts. This continuous planning improves the team’s chances of success as it incorporates new knowledge into the experience.

Scrum teams constantly respond to change so that the best possible outcome can be achieved. Scrum can be described as a framework of feedback loops, allowing the team to constantly inspect and adapt so the product delivers maximum value.

1.5. Suited for Large or Small Teams?

Large projects have high complexity and high risk of failure. There is lot of effort required to manage large projects and it requires lot of personnel. Communication and coordination overhead rises dramatically with team size. In the worst possible case where everyone on the project needs to communicate and coordinate with everyone else, the cost of this effort rises as the square of the number of people in the team.
Thus, Scrum is better off with small teams than large teams. Observations are that Scrum method performs efficiently for team sizes 7+/- 2.

Scrumalliance.org

1.6. Suited for Small or Large Projects?

Scrum is ideal for small teams of 5-9 members. Thus, it is ideal for small projects.
For large teams, rather than scaling, Scrum projects scale through having teams of teams. Although it's not the only thing necessary to scale Scrum, one well-known technique is the use of a "Scrum of Scrums" meeting. With this approach, each Scrum team proceeds as normal, but each team identifies one person who attends the Scrum of Scrums meeting to coordinate the work of multiple Scrum teams. These meetings are analogous to the daily Scrum meeting, but do not necessarily happen every day.

1.7. Advantages and Disadvantages

Advantages of Scrum:

- Improved productivity and measurable progress using burn-down graphs and daily Scrums.
- Changes can be easily incorporated as there are chances to reorganize and prioritize product backlog items between Sprints.
- Self-organizing and cross-functional teams.
- There is a shippable/ incremental component produced at the end of every Sprints. Due to this practice, there is continuous customer feedback. Product delivery can also be easily predicted.
- Due to regular builds, bugs can be identified at an early stage of the development cycle.
- Regular releases and sprint reviews and retrospectives make the teams very productive and there is continuous product and process improvement.

Disadvantages of Scrum:

- The Scrum method is not realistic and it does not scale very well.
- The person who drives the project, the Scrum Master should be experienced and technically sound to lead and guide the developers.
- The Sprints and meetings in Scrum have a very time-boxed approach. But sometimes, it may occur that the things don't get completed due to reprioritization or some change in requirements or other outside factors. Additional sprints may be necessary in these cases which could add to overall time.
- Developers must be knowledgeable to be able to follow fast paced Sprint meetings and time-boxed work periods.
- There is very little or almost no documentation during the development cycle in Scrum.
- Tasks should be clearly defined in Product and sprint backlogs and the definition of "Done" should be consistent with each member of the team.

----

2. Lean

2.1. Description

The Lean Philosophy derives from the Toyota Production System was developed to improve the car production of Japanese car manufacturer Toyota. It is said to be one of the reasons why Toyota became a market leader in the automobile sector. The focus of the Toyota Production System lies in the "absolute elimination of waste" This is achieved by two core ideas:

- Just In Time Production means only to make and supply what is needed at the moment. Both over- and underproduction are avoidable waste.

- Autonomation means the automatic detection of defects as early as possible. Production may only continue when the defect is resolved. No defective product shall go to the next step in production. No defective machine all continue to produce.

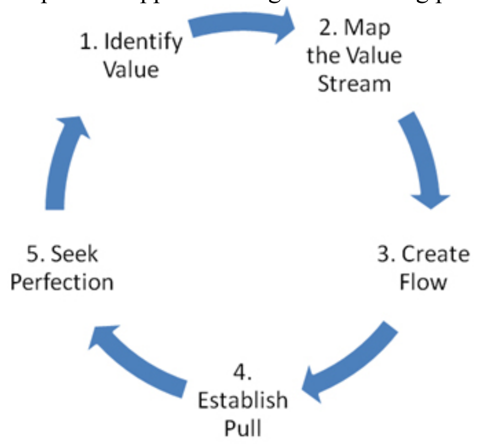
2.2 Detailed Process

Lean philosophy is based on 7 Principles:

- Optimize the whole
- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team

- Build integrity in

These principles are applied during the following process:



2.1.1. Determining value
Value is determined by customer's requirement for a software product or service.

2.1.2. Value Stream
Once the value has been determined, the next step is mapping the "value stream," or all the steps and processes involved in taking a specific product from raw materials and delivering the final product to the customer.
Value-stream mapping is a simple but eye-opening experience that identifies all the actions that take a product or service through any process.
The idea is to draw, on one page, a "map" of the flow of product through the process.
The goal is to identify every step that does not create value and then find ways to eliminate those wasteful steps. In software development waste can be dead code, resources that are not assigned any work, a requirement that has changed and doesn't need any development, delay in delivery, change in processes followed.

2.1.3. Flow and Set Based Development
After the waste, has been removed from the value stream, the next step is to be sure the remaining steps flow smoothly with no interruptions, delays, or bottlenecks.
For this in Lean, instead of choosing one design as per customer requirements, the team makes several designs prototypes that satisfy the requirements. At early stages, they spent time in exploring various alternatives designs so that customer can get clarity on how the final product will be at very early stage of development. The team then eliminates alternatives as per customer feedback, finalizes on one prototype and goes on refining it till it reaches the final stage.

2.1.4. Pull Systems
With improved flow, delivery time can be improved. This makes it much easier to deliver products as needed, as in "just in time" manufacturing or delivery. This means the customer can "pull" the product from you as needed. Thus, products don't need to be built in advance or materials stockpiled, creating expensive inventory that needs to be managed, saving money for both the manufacturer/provider and the customer. Pull systems are implemented using Kanban Boards.

2.1.5. Perfection
As gains continue to pile up, it is important to remember lean is not a static system and requires constant effort and vigilance to perfect. Every employee should be involved in implementing lean.

2.2. Management Structure / The Scrum Team

Lean software development, places software development as a step in a product value stream, regards developers as members of a larger product team, and expects all product team members to become engaged in the overall success of the product. There's no product owner or customer roles in lean development.
A chief architect has overall responsibility for a complete product, including its success in the marketplace, and engages every-one on a multidiscipline product team in delivering that success. There's no intermediate role prioritizing work for a separate software development team.

2.3. Alignment with Agile Manifesto

- Individuals and interactions over processes and tools

  Lean methodology regards developers as members of a larger product team, and expects all product team members to become engaged in the overall success of the product. Thus, collaboration is of utmost importance. There are no defined roles and no rigid processes. Thus, group members work together to achieve the final goal.

- Working software over comprehensive documentation

  The main motive of Lean is to eliminate waste and optimize overall process towards achieving customer satisfaction. Instead of locking down to one design the teams make several designs at early stages and modify and choose the ones that satisfy the customer's requirements. Thus, there is continuous refinement of working software and regular releases.

- Customer collaboration over contract negotiation

Customer is given utmost importance. Value is determined by customer's requirement for software products or service.

- Responding to change over following a plan

  Lean uses kanban which provides easy way to introduce changes through the pull system. As a result, the team is constantly trying to adapt to the customer's changing requirements. Therefore, the team is flexible in adapting to the needs of the clients at any point in the software development life cycle.

2.4. Suited for Large or Small Teams?

Lean is suited for small teams. As Lean methodology eliminates waste through such practices as selecting only the truly valuable features for a system, prioritizing those selected, and delivering them in small batches. It emphasizes the speed and efficiency of development workflow, and relies on rapid and reliable feedback between programmers and customers. Lean uses the idea of work product being "pulled" via customer request. It focuses decision-making authority and ability on individuals and small teams, since research shows this to be faster and more efficient than hierarchical flow of control. Lean also concentrates on the efficiency of the use of team resources, trying to ensure that everyone is productive as much of the time as possible. It concentrates on concurrent work and the fewest possible intra-team workflow dependencies.

2.5. Suited for Small or Large Projects?

Lean software development works for small as well as large projects. It is more suitable for small projects as it is easy to manage waste within a small team. Also, it is easier to maintain focus with lesser tasks to be done. Typically, large projects can be broken down as multiple small projects and lean methodology can be applied to each individual small project so as a whole the entire large project can be made lean.

2.6. Advantages and Disadvantages

Advantages:

- The elimination of waste leads to the overall efficiency of the development process. This is turn speeds up the process of software development which reduces project time and cost.
- Delivering the product early is a definite advantage. It means that the development team can deliver more functionality in a shorter period, hence enabling more projects to be delivered.
- Empowerment of the development team helps in developing the decision-making ability of the team members which in turn, creates a more motivated team.

Disadvantages:

- The project is highly dependent on cohesiveness of the team and the individual commitments of the team members.
- Success in the project depends on how disciplined the team members are and how exceptional are their technical skills.
- The project sponsors and clients need to know what they want and make decisions they will stick to. In lean software development, these decisions can be made later than say when using waterfall methodologies, which should be an advantage. And in lean the whole objective of using this over say agile methodology is to enable your development to be done faster and cheaper than would otherwise be possible. This of course means decisions must be made promptly when required and stuck to.
- Kanban boards should be updated constantly so that there are no outdated tasks on the board.
- In lean you allow the software requirements specification (SRS) to evolve. This can cause problems. Flexibility is great, but too much of it will quickly lead to a development which loses sight of its original objectives and which never finishes.

----

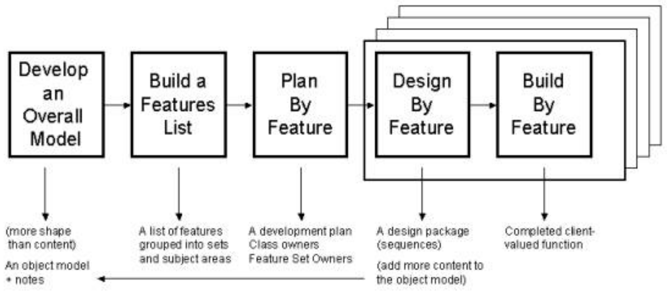
3. Feature Driven Development

3.1. Description

Feature driven development combines key advantages of key agile approaches like Scrum and extreme programming. Some of its plus features include dynamic feature teams, peer reviews, and just enough design initially (JEDI). FDD scales easily to much larger teams and projects than generally recommended for other agile approaches. When done well, FDD produces meaningful, accurate and timely status reporting and progress tracking for all levels of project leadership and can prove most satisfying way of working in a software development team.

3.2. Detailed Process

FDD is a 5 step development process:


Develop an Overall Model
Build a Features List
Plan By Feature
Design By Feature
Build By Feature
(more shape than content)
An object model + notes
A list of features grouped into sets and subject areas
A development plan
Class owners
Feature Set Owners
A design package (sequences)
(add more content to the object model)
Completed client-valued function

3.2.1 Develop an Overall Model

- In FDD, a team invests just enough effort at the beginning of a project in exploring the structure of the problem by building an object model of the problem domain.
- The model has just enough detail to form a good shared understanding, vocabulary and conceptual framework for the project.
- In addition, the modelling activity uncovers just enough knowledge to build a good product backlog (called a Feature List in FDD) and initial, overall release-level project plan.

3.2.2 Build Feature List

- Using the knowledge gained during modelling process, the team next constructs the FDD equivalent of an initial, overall product backlog.
- Instead of user stories or backlog items, FDD talks about features.
- In FDD, a feature is a small piece of client-valued function expressed in the form: <action> the <result> <by/for/of/to> a(n) <object>;
  for example, 'calculate the total of a sale'.

3.2.3 Plan by Feature

- The third activity in FDD, is sequencing the sets of features for activities into a high-level plan and assign them to chief programmers.
- Developers are also assigned to own classes identified in the overall object model.
- Again, this differs from many Agile methods that have the concept of collective ownership of code, and when combined with FDD's dynamically-formed feature teams, it provides an alternative solution to the problems that collective ownership seeks to resolve.

3.2.4 Design by Feature

- Using the knowledge gained during and since the modelling activity, a lead developer (called a Chief Programmer in FDD terms) selects the small group of features that make most sense to develop over the next few days.
- The chief programmer identifies the domain classes likely to be involved, and the corresponding class owners form the feature team for this work.
- Depending on circumstances, others with specific skills such as testers, technical authors and user experience designers may join the team.
- This feature team works together with the help of a domain expert to analyze the details of each selected feature and design the solution for each.

3.2.5 Build by Feature

- The team then work individually on the resulting coding and testing tasks, holding a code inspection when all is done.
- Once the chief programmer is satisfied that the work is complete, the completed features are promoted to the main, regular build, the feature team is disbanded, and the two processes are repeated for another small set of features.

3.3. Management Structure / The FDD Team

3.3 Project Manager

He/she is the administrative lead of the project responsible for reporting progress, managing budgets, fighting for headcount, managing equipment, space and resources etc.

3.3.1 Chief Architect

- He/she is responsible for the overall design of the system.
- In FDD, he/she is essentially responsible for facilitating the design of the system through collaborative working sessions.
- They are, however, responsible for guarding the conceptual integrity of the design ensuring individual feature designs do not compromise it.
- This is a deeply technical role requiring excellent technical and modelling skills and good facilitation skills.
- The Chief architect resolves disputes over design that the chief programmers cannot resolve themselves.
- He or she steers the project through the technical obstacles confronting the project.

3.3.2 Development Manager

He/she has the ultimate say on the day-to-day developer resourcing conflicts within the project and steers the team through potential resource deadlock situations.

3.3.3 Chief Programmers

- They are experienced developers that have been through the whole software development lifecycle a few times.
- They participate in the high-level requirements analysis and design activities of the project and are responsible for leading small teams of 3-6 developers through low-level analysis, design and development of the new software’s features.
- The team works collaboratively on the analysis and design for each feature.

- The Chief Programmer facilitates this work but also makes the final call if the team cannot decide between two options.

3.3.4 Class Owners

- They are developers that work as members of small development teams under the guidance of a chief programmer to design, code, test, and document the features required by the new software system.
- Class owners are often talented developers who, with more experience, will become able to play chief programmer roles in the future, or they are talented developers who are content to be master programmers and want nothing to do with leading or managing other people.

3.3.5 Domain Experts

- These are users, clients, sponsors, business analysts or any mix of these.
- They use their deep knowledge of the business to explain to the developers in various levels of detail the tasks that the system must perform.

3.4. Alignment with Agile Manifesto

- Individuals and Interactions over processes and tools

  FDD has concept of dynamic feature teams. They are built to work on features as per the expertise of the members and dynamically realigned. This essentially means that there can be no single process as the teams are dynamic in nature. This necessitates good interactions in order to achieve success in product delivery.

- Working Software over comprehensive documentation

  There are regular builds of features. Some teams build weekly, others daily and others continuously. This allows for early detection of integration problems and makes sure that there is always something to demo to the client.

- Customer collaboration over contract negotiations

  FDD uses the concept of JEDI – Just enough design initially. Once it is approved the teams begin working and refine the features as per regular customer feedbacks.

- Responding to change over following a plan

  FDD keeps the iterations involved in development short in terms of time. The process in general is very light weight. Thus, it is easier for teams to respond to changes in the requirements and accommodate them into the plan for next iteration without introducing much delays.

3.5. Suited for Large or Small Teams?

A feature team is a small, dynamically formed team that develops a small activity. By doing so, multiple minds are always applied to each design decision and multiple design options are always evaluated before one is chosen. Thus, FDD is suited for small teams.

3.6. Suited for Small or Large Projects?

As projects with small scope don't have many features to work on; you cannot build a feature list and follow feature driven development approach. Thus, it is not suitable for small projects.
FDD helps to move larger size projects and obtain repeatable success. The nature of FDD of dynamically forming feature teams and working on the different features is very much suited for large projects. For a large project, 'n' small teams can work on 'n' features.

3.7. Advantages and Disadvantages

Advantages:

- There is a concept of individual code ownership in FDD, which promotes a sense of pride and also encourages expertise in a area.
- There are regular builds in FDD, which lead to detection of problems at an early stage. Also the team has always something ready to show to the client along with detailed documentation.
- In FDD, there are regular code inspections in Build by Feature phase by which code integration problems are resolved periodically.
- The concept of JEDI, Just Enough Design Initially, avoids refactoring as clients are presented with minimal design before start of actual development.
- As the multiple teams work on different features, it is very easy to track the overall progress of the project by tracking how many or how much of the features are completed.

Disadvantages:

- The concept of individual code ownership has too much risk associated with it, in case if that class owner leaves the team, there is no one who knows his class/module.
- For developers to be class owners in feature teams, they must have good technical knowledge and they must be experienced. Fresher's cannot fit best in this system.
- This is can be used but it is not ideal for smaller projects. As project is broken down into features

and then developed, small project might not have many features and this model could fail in that case.

---

4. Comparison

Agile Principle	Scrum	Lean	FDD
Individuals and interactions over processes and tools	Scrum is a team-based effort to delivering value to the business by working in full collaboration.	There are no defined roles and no rigid processes. Thus, group members work together to achieve the final goal.	FDD has concept of dynamic feature teams. There can be no single process as the teams are dynamic in nature.
Working software over comprehensive documentation	At end of every Sprint, Scrum requires a working, finished product increment.	Instead of locking down to one design the teams make several designs at early stages and modify and choose the ones that satisfy the customer's requirements. Thus, there is continuous refinement of working software and regular releases.	There are regular builds of features. Some teams build weekly, others daily and others continuously. As a result, the finished iteration itself serves as document.
Customer Collaboration over contract negotiation	The team, especially the product owner, collaborates with stakeholders to inspect and adapt the product vision so the product will be as valuable as possible.	Value is determined by customer's requirement for software products or service.	FDD uses the concept of JEDI – Just enough design initially. Once it is approved the teams begin working and refine the features as per regular customer feedbacks.
Responding to Change over following a plan	The team's goal is not to blindly follow the plan; the goal is to create value and embrace change. The thought process and ideas necessary for planning are more important than the plan itself.	Lean uses Kanban which provides easy way to introduce changes through the pull system. As a result, the team is constantly trying to adapt to the customer's changing requirements.	FDD keeps the iterations involved in development short in terms of time and light weight. This makes way to easily adopt to change.

Acknowledgment

This paper is written under guidance of Prof. Richard Ens of Stevens Institute of Technology. His deep insights into the field of SDLC and Agile development have been of tremendous help in completion of this paper.

References

[1] *"Product Owner"*, The Scrum Guide
[2] *"Scrum Master",* https://www.mountaingoatsoftware.com/agile/scrum/scrummaster, http://scrummethodology.com/the-scrummaster-role/
[3] "*Scrum Team size"*, https://www.scrumalliance.org/
[4] "*Advantages and Disadvantages*", https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban
[5] *"Scrum of scrums"*, https://www.mountaingoatsoftware.com/agile/scrum/team
[6] *"Lean description"*, http://www.freenerd.de/assets/bachelors_thesis_johan_uhle.pdf
[7] "Lean Process", https://www.asme.org/engineering-topics/articles/manufacturing-design/5-lean-principles-every-should-know
[8] *"Lean Team Size"*, https://www.versionone.com/agile-101/agile-methodologies/
[9] *"Lean Disadvantages"*, http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-lean-software-development.html

[10] *“Description FDD”*, http://www.step-10.com/SoftwareProcess/FeatureDrivenDevelopment/

[11] *“FDD large or small teams”*, https://en.wikipedia.org/wiki/Feature-driven_development

[12] *“Feature Driven Development Process”,* http://www.step-10.com/SoftwareProcess/FeatureDrivenDevelopment/

[13] *“FDD Advantages”*, http://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages/