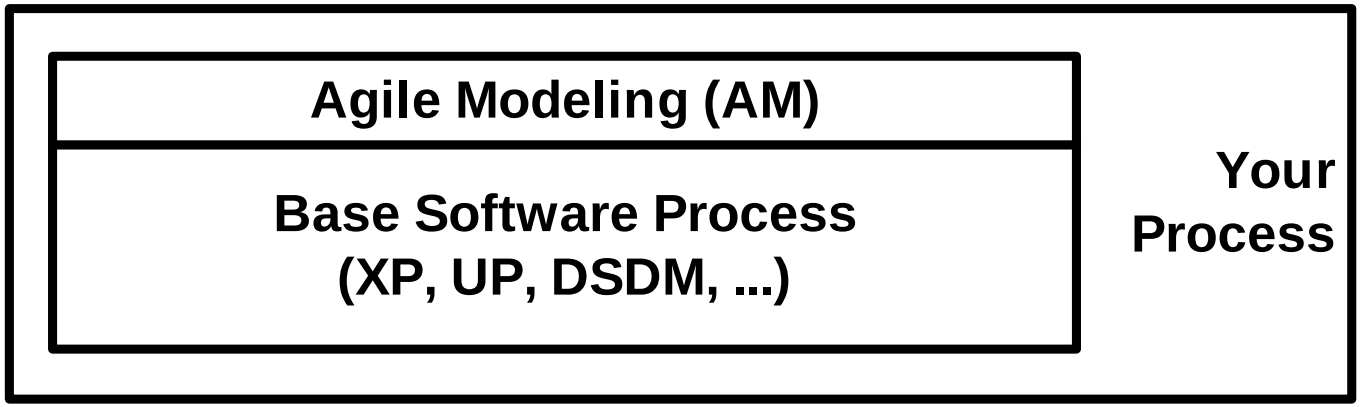
Agile Modeling

Lecture Objectives

- Specifics of architecting software in agile manner

Agile Modeling

Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of software-based systems



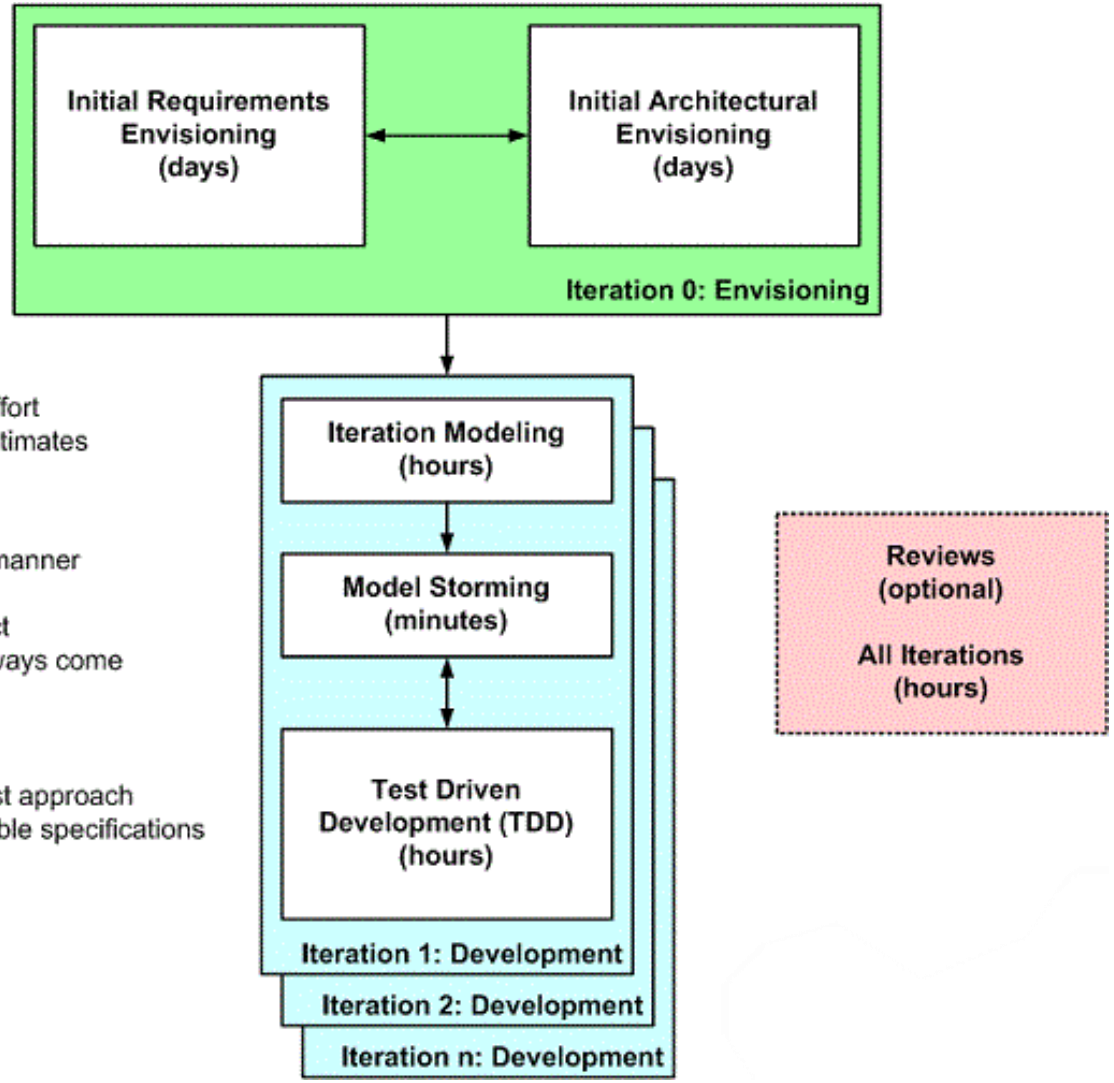
Agile Model Driven Development

- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision

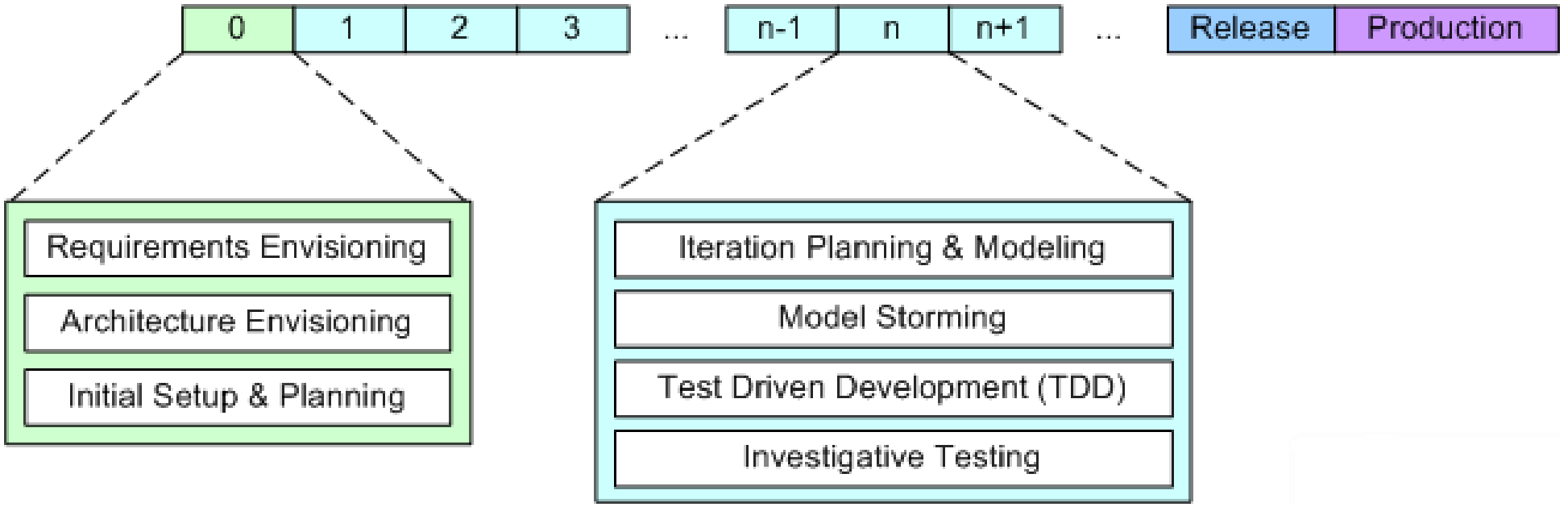
- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

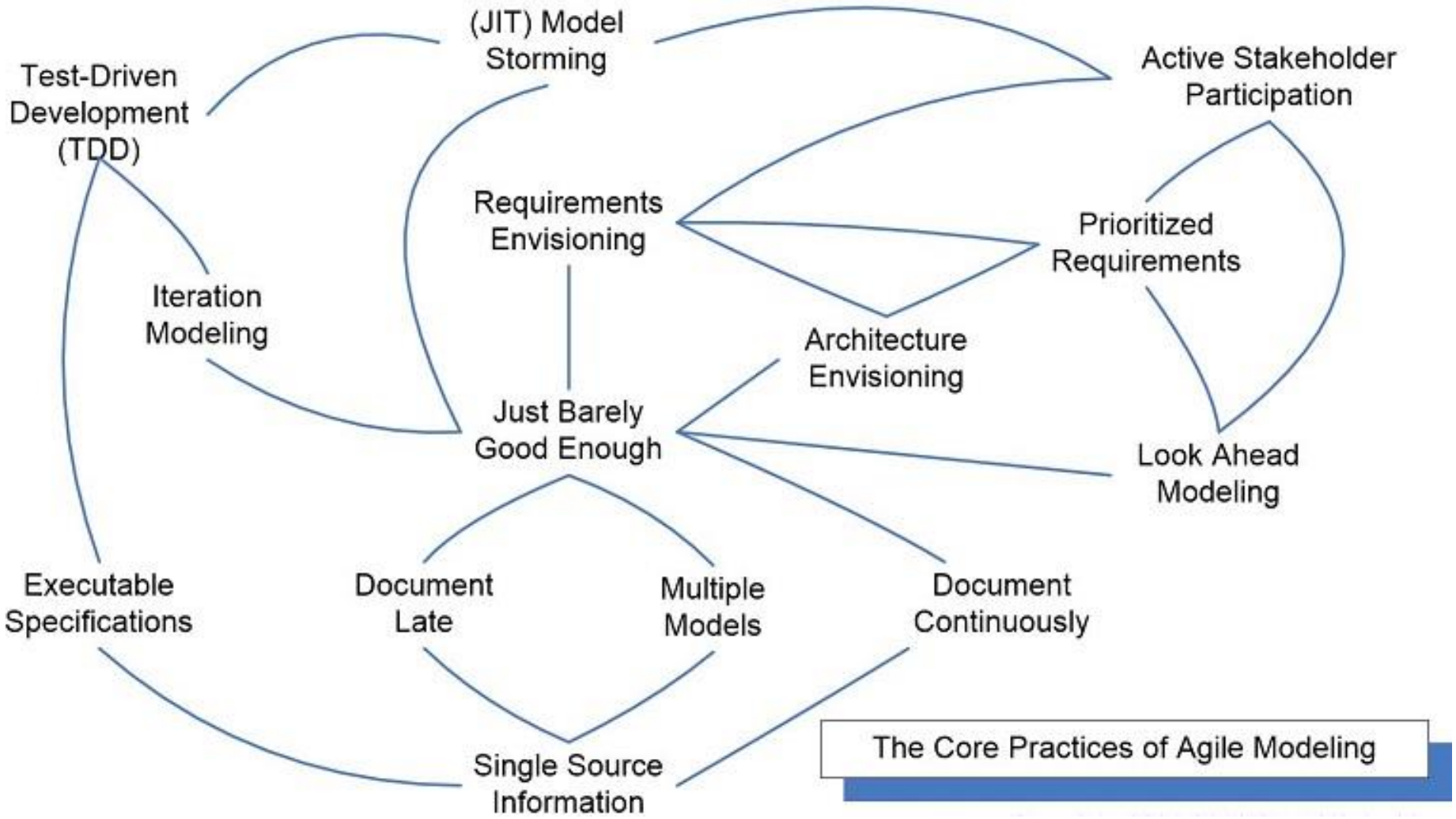
- Develop working software via a test-first approach
- Details captured in the form of executable specifications



Agile Model Driven Development



Practices of AM



Active Stakeholder Participation

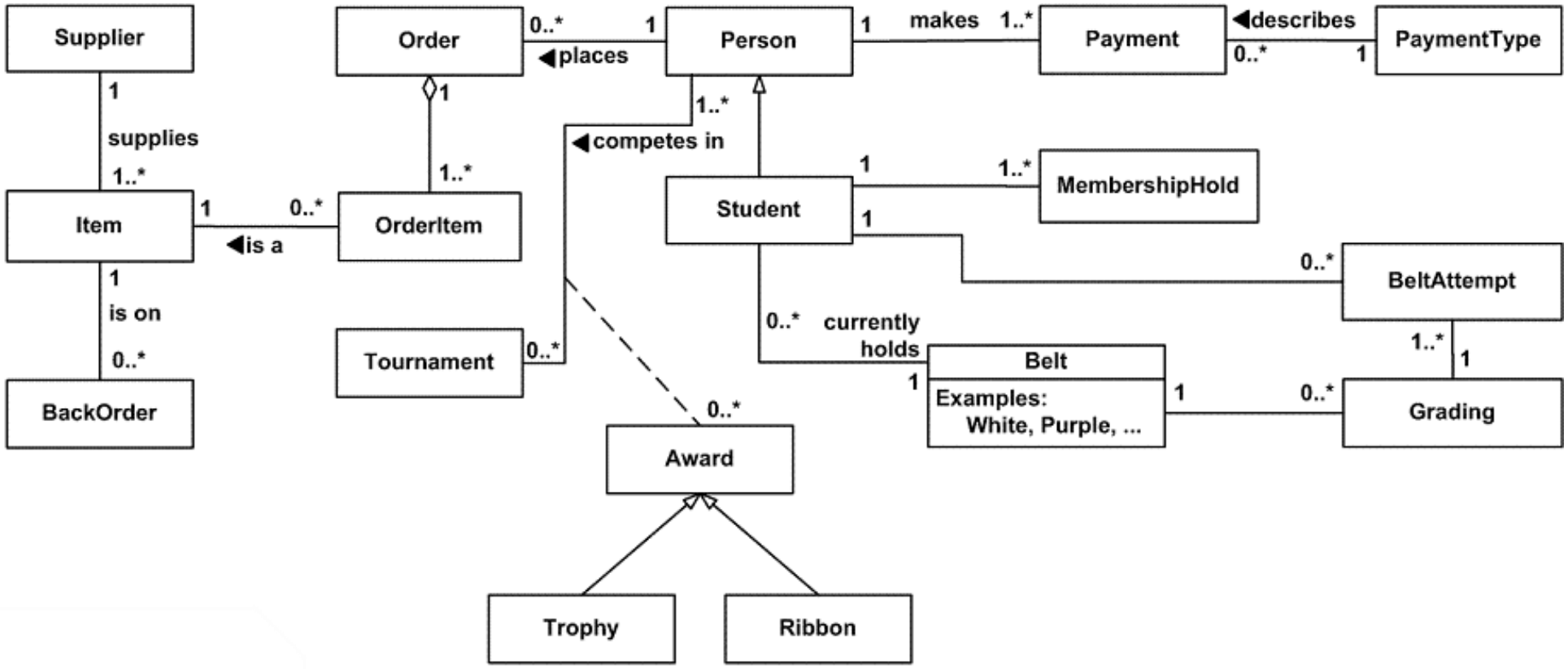
1. **Timely decisions**. Stakeholders must be prepared to share business knowledge with the team and to make both pertinent and timely decisions regarding project scope and requirement priorities
2. **Inclusive modeling**. You need to adopt models which stakeholders can easily learn and adopt
3. **Management requires IT skill and knowledge**. For senior managers to effectively support your project they must understand the technologies and techniques that your team is using
4. **Production staff are involved from the start**. Your support staff must take the time to learn the nuances of your system and/or your team will need to provide them with training
5. **Take an enterprise view**. You need to work with other project teams if your system needs to integrate with other systems

Requirements Envisioning

- Performed during Iteration 0 of an agile project
- For the first release of a system you need to take several days to identify some high-level requirements as well as the scope of the release (what you think the system should do)
- The goal is to understand the requirements at a high-level, it isn't to create a detailed requirements specification early in the lifecycle
- May have some form of:
  - Usage model
  - Domain model
  - User interface model(s)

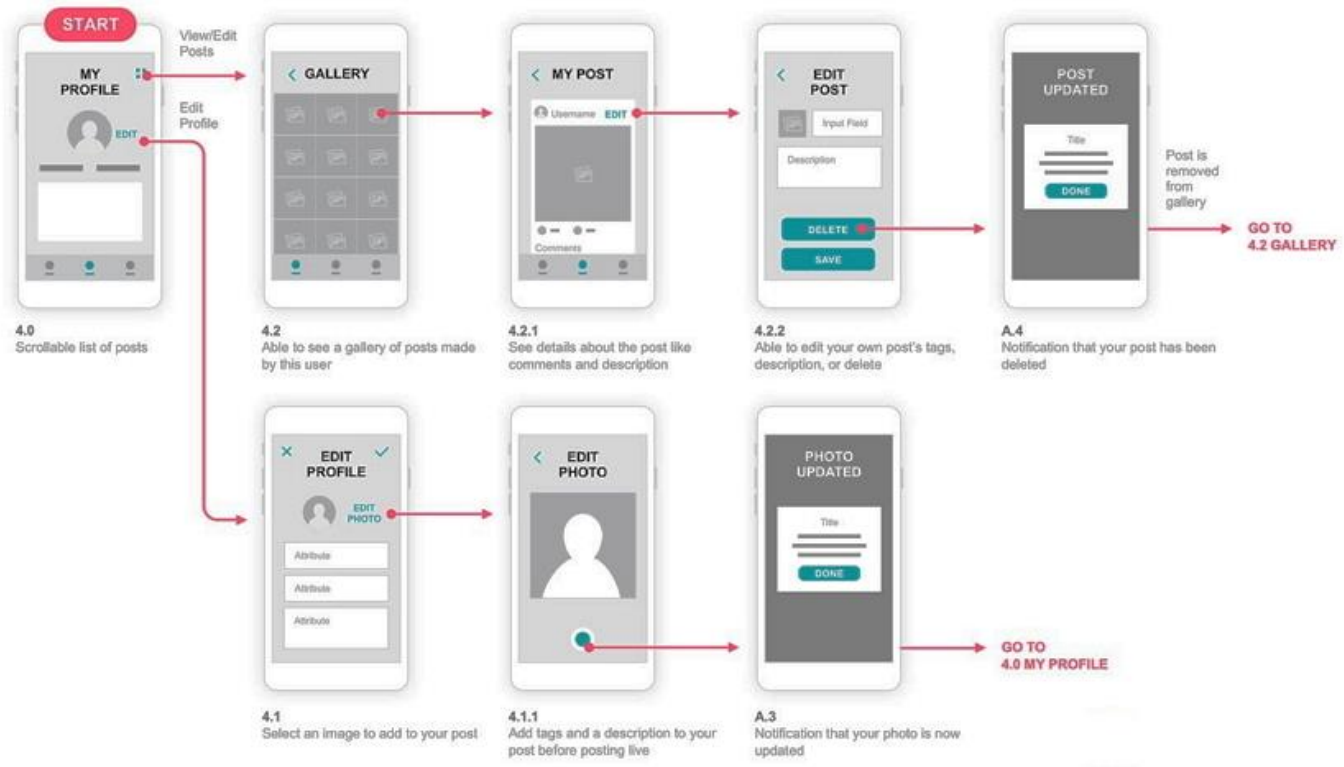
Domain model

- Capturing the main business entities and the relationships between them



User interface flow

- Explore how you will architect the UI of your system by looking at the flow between major UI elements, including screens/pages, reports, etc.
- Critical to the system's success because the user interface is the system to your stakeholders



Architecture Envisioning - 1

- Early in the project you need to have at least a general idea of how you're going to build the system
- This modeling work is based on and performed in parallel to requirements envisioning
- Your architecture will evolve over time, so it does not need to be very detailed yet (just good enough), and very little documentation (if any) needs to be written

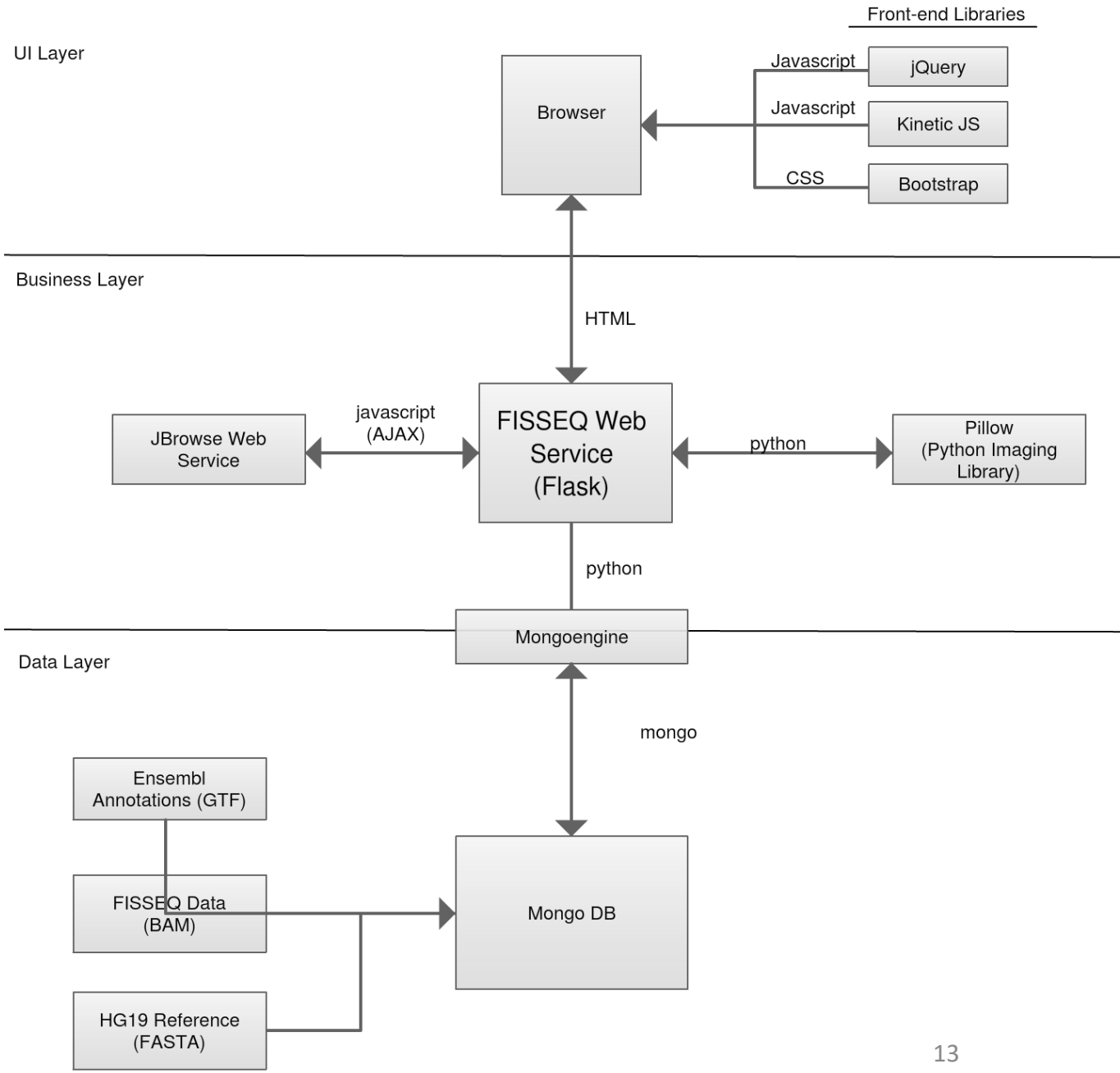
Architecture Envisioning - 2

- When doing initial architectural modeling focus on high-level free-form diagrams which overview the idea of how the system will be built

- Typically the focus is on:
  - Technology diagrams
  - User interface (UI) flow
  - Domain models
  - Change cases

Technology diagrams

- Could be technology stack or deployment diagram
- These diagrams are useful because they depict the major software and hardware components and how they interact at a high level



Change case

- Used to describe new potential requirements for a system or modifications to existing requirements
- Describe the potential change to your existing requirements, indicate the likeliness of that change occurring, and indicate the potential impact of that change

Template

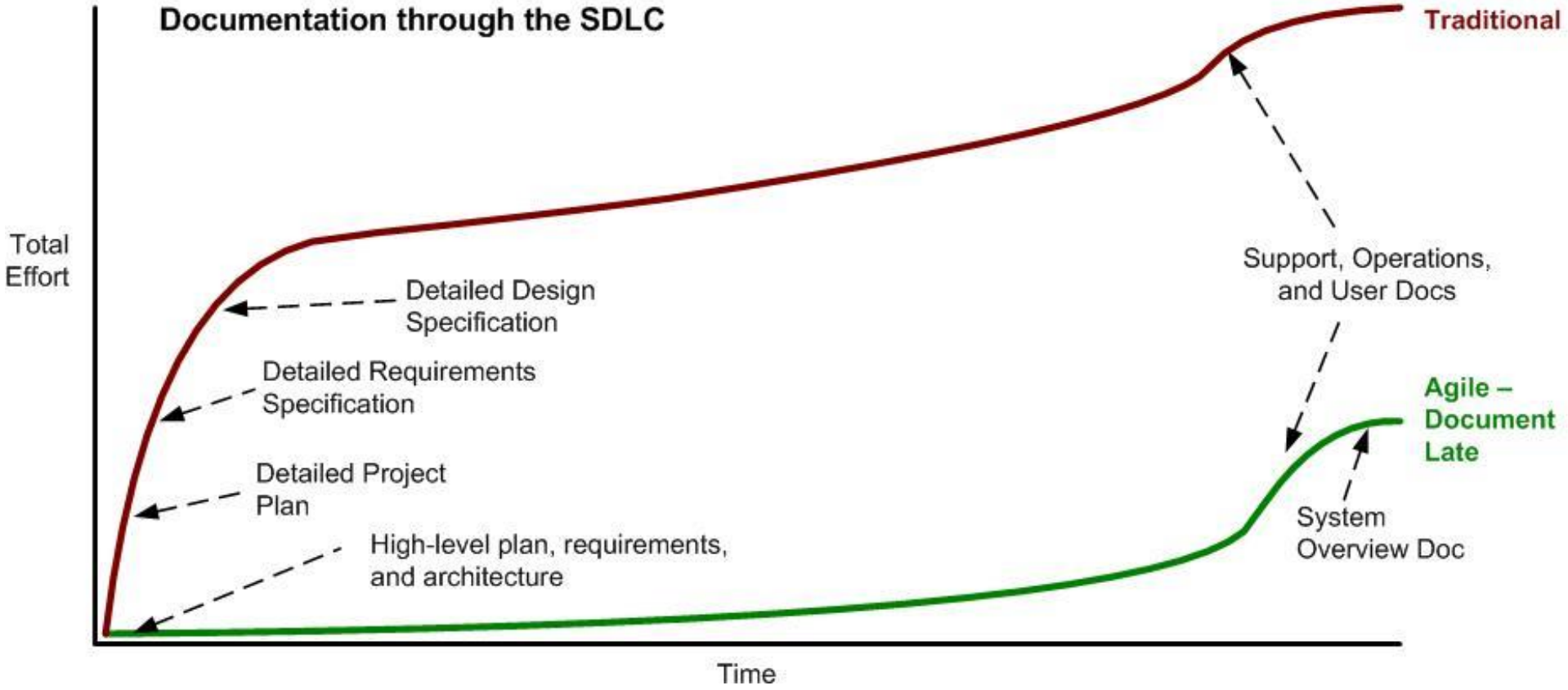
Name: (*One sentence describing the change*)
Identifier (*A unique identifier for this change case, e.g. CC17*)
Description (*A couple of sentences or a paragraph describing the basic idea)*
Likelihood (*Rating of how likely the change case is to occur (e.g. High, Medium, Low or %)*)
Timeframe (*Indication of when the change is likely to occur*)
Potential Impact (*Indication of how drastic the impact will be if the change occurs*)
Source (*Indication of where the requirement came from*)
Notes

Document Continuously

- Wait for the information to stabilize. Write the documentation after you've done the majority of the development work. If you document information that isn't yet stable, you run the risk of having to rework your documentation
- Write documentation during the iteration with long iterations
- Write documentation the following iteration with short iterations

Document Late

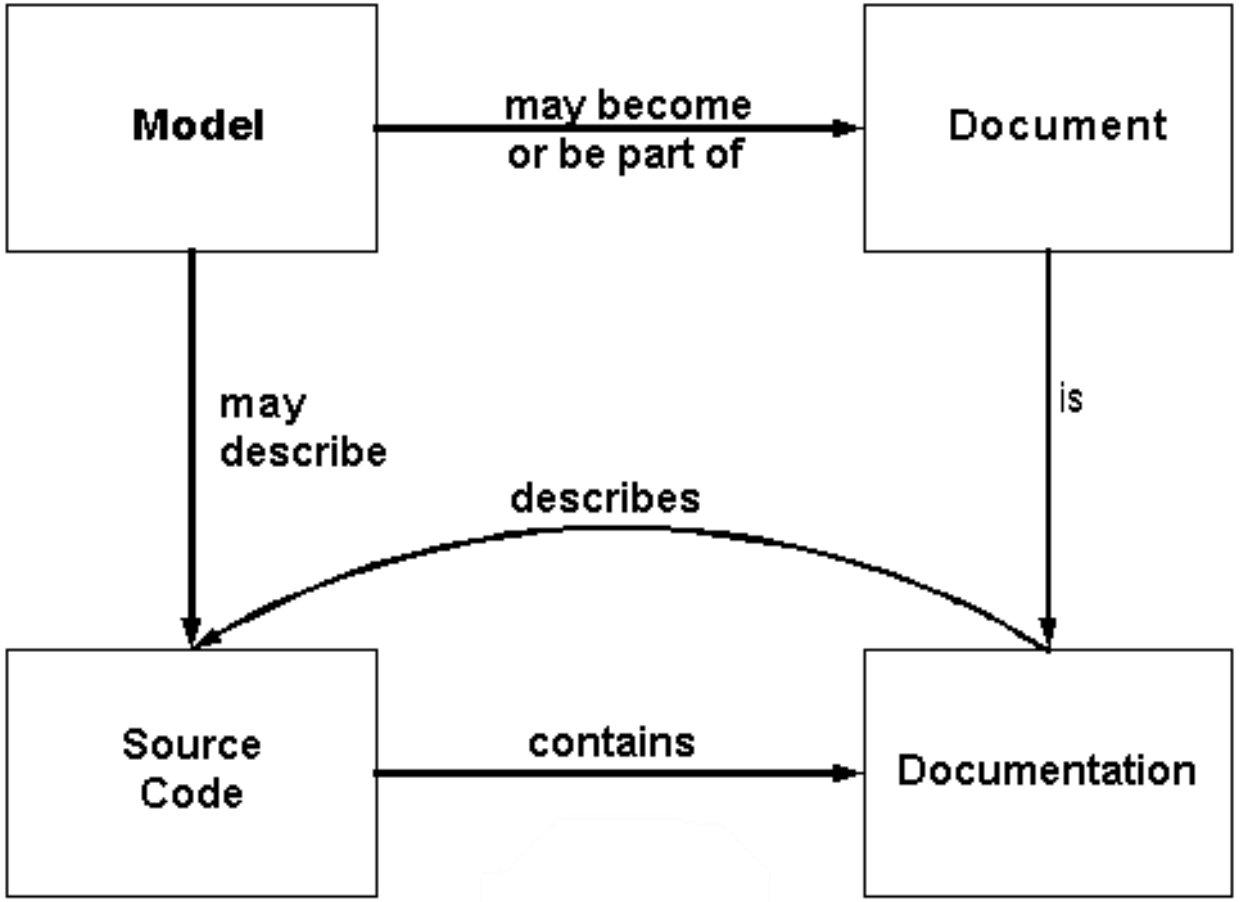
Create of all deliverable documentation as late as possible, creating them just before you need to actually deliver them



Executable Specifications

- Most programmers don't read the written documentation for a system, instead they prefer to work with the code
- Well-written unit/developers tests provide a working specification of your functional code – and as a result unit tests effectively become a significant portion of your technical documentation
- Similarly, acceptance tests can form an important part of your requirements documentation. Your acceptance tests define exactly what your stakeholders expect of your system, therefore they specify your critical requirements

Models, Documents, and Source Code



Iteration Modeling - 1

- At the beginning of each iteration an agile team will typically plan (estimate and schedule) the work that they will do that iteration.

- To estimate each requirement accurately you must understand the work required to implement it, and this is where modeling comes in. You discuss how you're going to implement each requirement, modeling where appropriate to explore or communicate ideas.

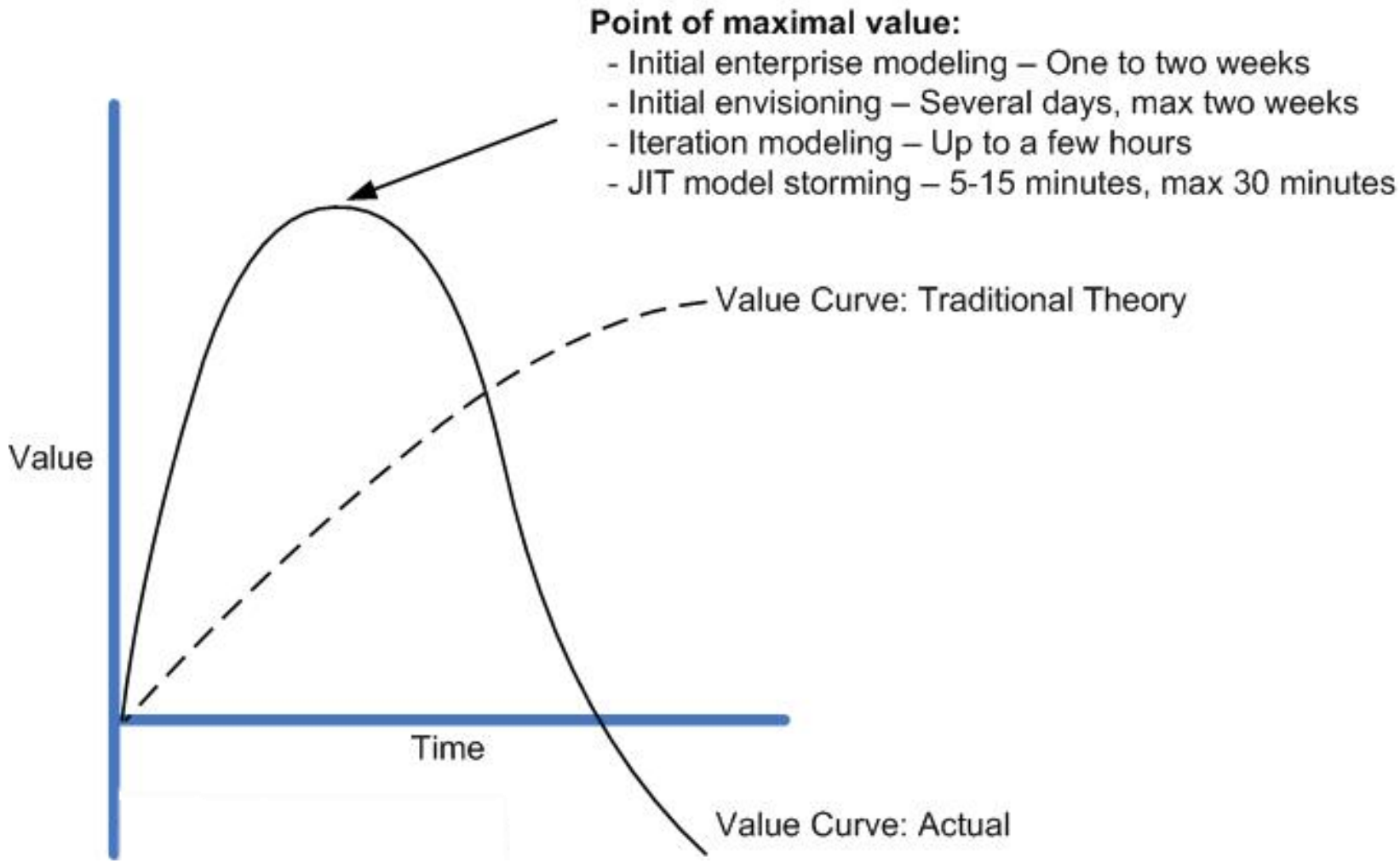
- This modeling in effect is the analysis and design of the requirements being implemented that iteration.

Iteration Modeling - 2

- Remember, the goal at this point in time is to simply plan the work

- If more details are required, they can be identified:
  - through more detailed model storming throughout the iteration
  - in the form of tests via a test-driven development (TDD) approach

Just Barely Good Enough Artifacts



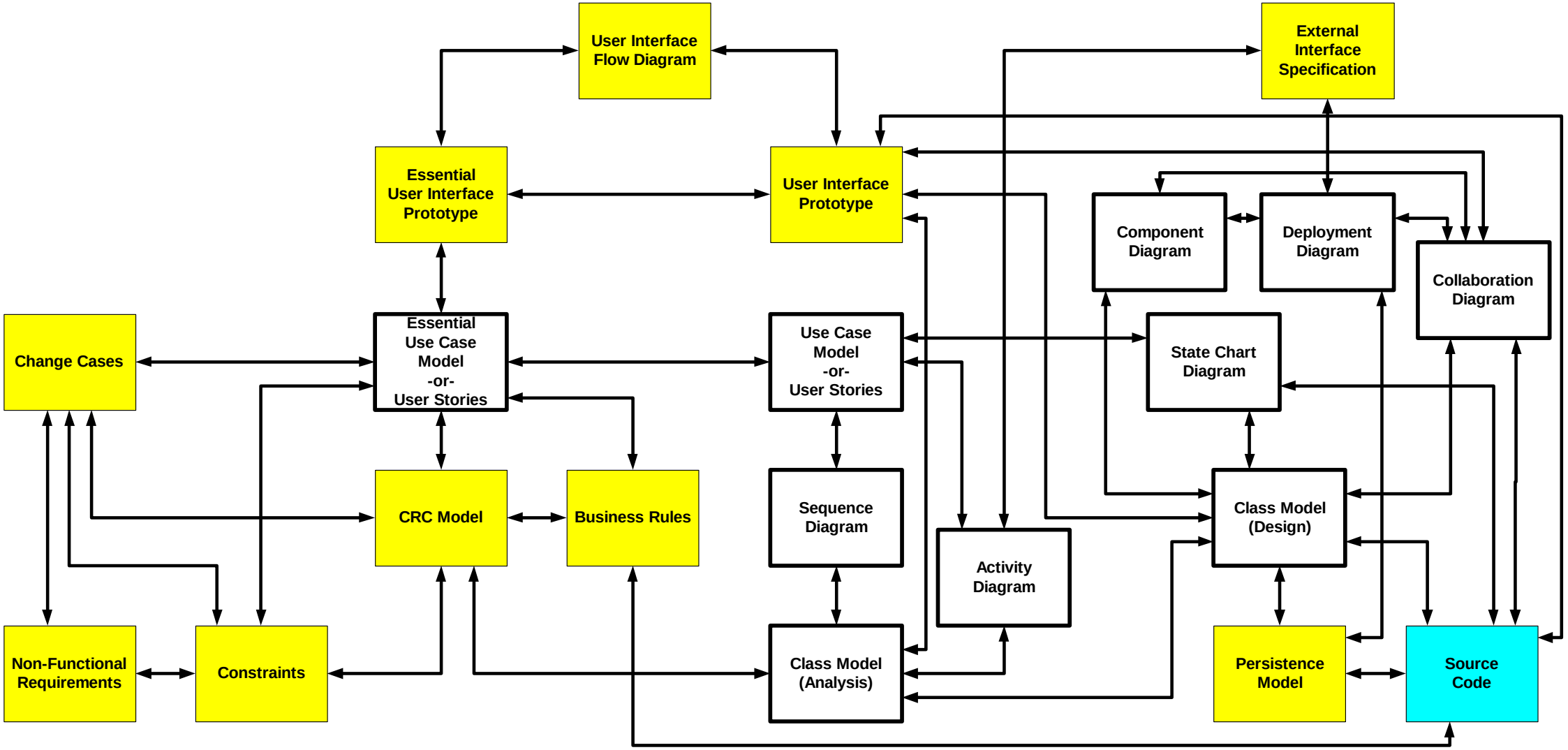
Model Storming

- Model storming is just in time (JIT) modeling
- These "model storming sessions" are typically spontaneous events, lasting for five to fifteen minutes
- The people get together, gather around a shared modeling tool (e.g. the whiteboard), explore the issue until they fully understand it, then they continue on
- By waiting to analyze the details JIT, you have much more domain knowledge than if you had done so at the beginning of a project

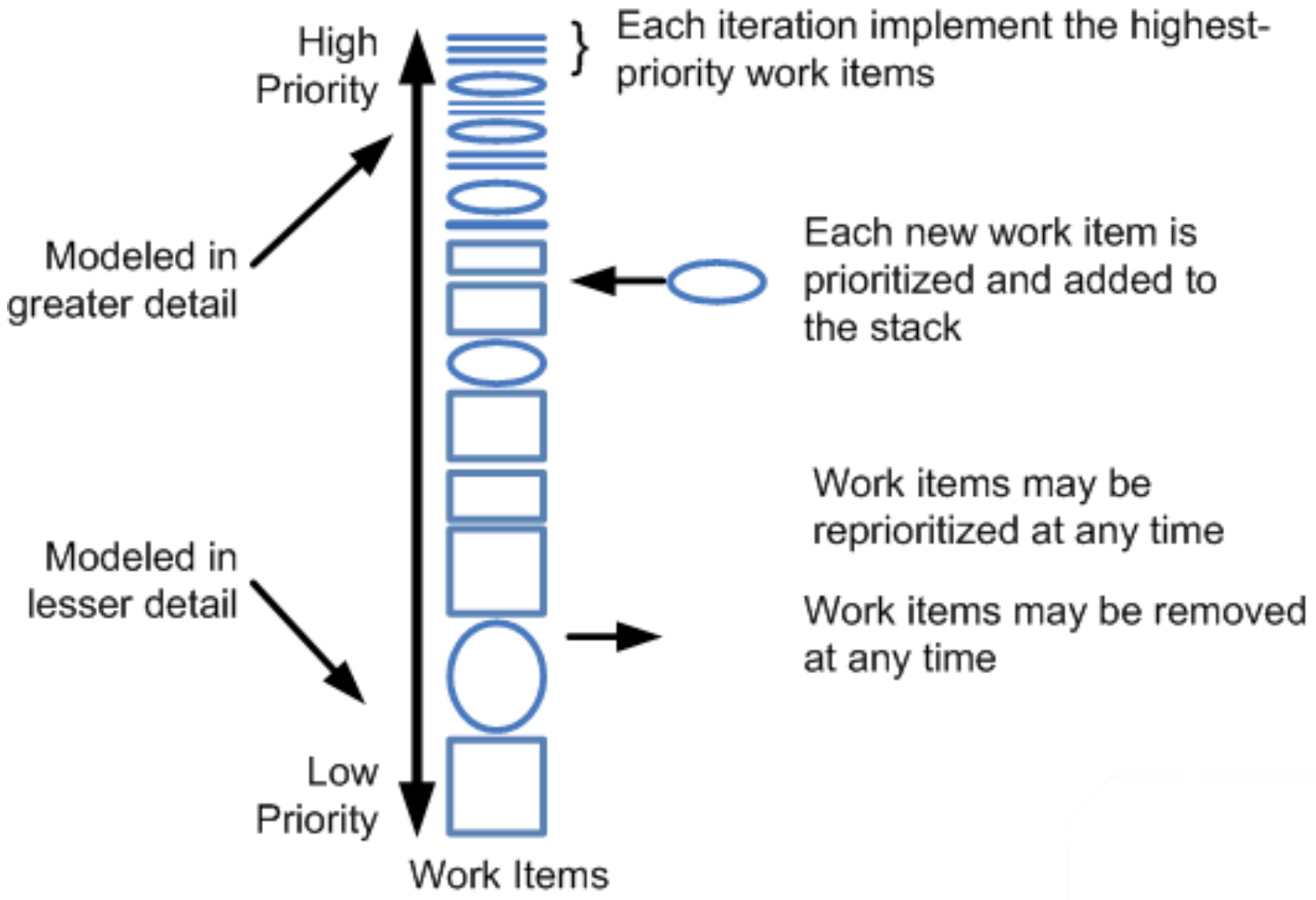
Look Ahead Modeling

- Sometimes you find yourself in a situation where you can't easily model storm a design issue for a few minutes on a just-in-time (JIT) basis
- With look ahead modeling your team does just enough modeling to understand a requirement which has been assigned to some point in the near future
- To remain agile you model just the complicated requirements of the coming iteration, not every single one

Multiple Models

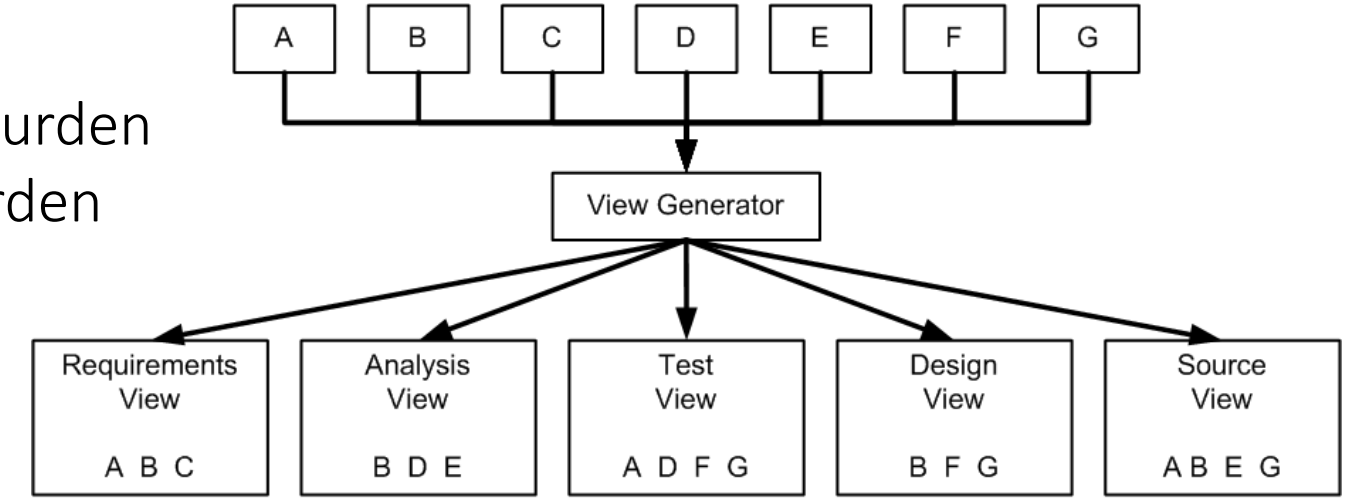


Prioritized Requirements



Single Source Information

- When you are modeling you should always be asking the questions "Do I need to retain this information permanently?", "If so, where is the best place to store this information?" and "Is this information already captured elsewhere that I could simply reference?".
- It helps you to:
  - Reduce your maintenance burden
  - Reduce your traceability burden
  - Increase consistency

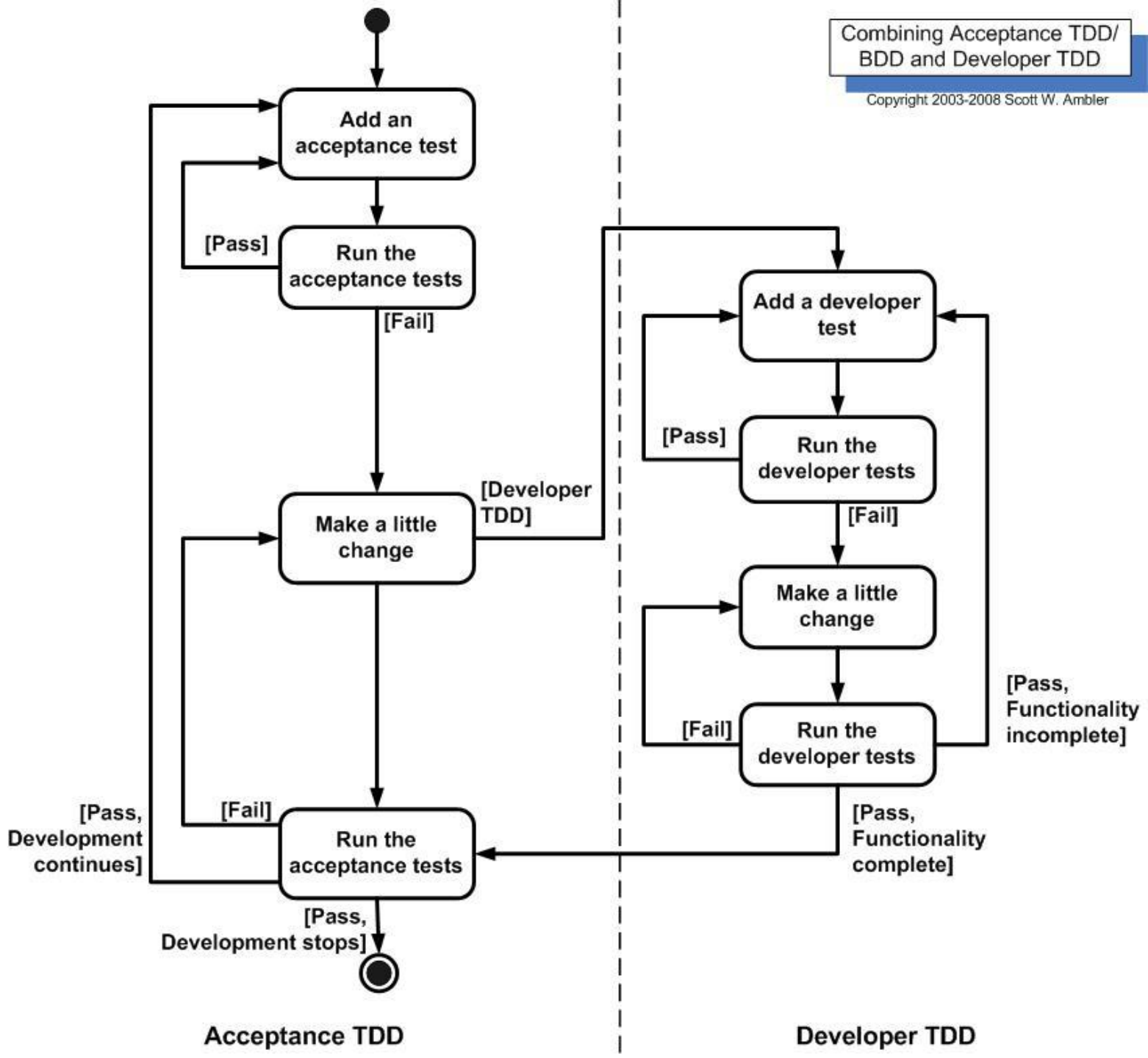


Test-Driven Design (TDD) - 1

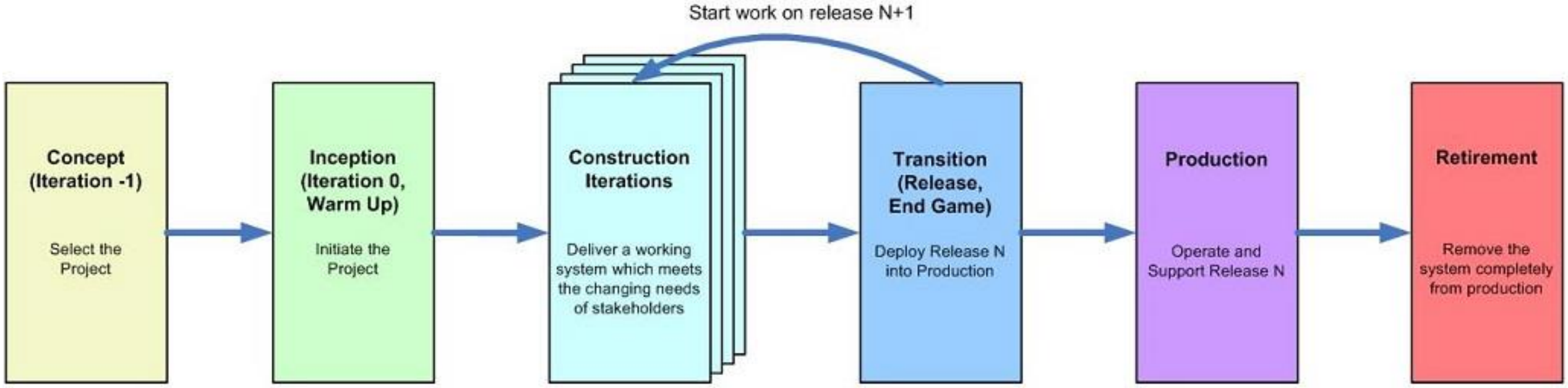
- Test-driven development (TDD) is an evolutionary approach to development which combines test-first development and refactoring

- When you first go to implement a new feature, analyze whether the existing design is the best design possible to implement that functionality
  - If so, you proceed via a TFD approach
  - If not, you refactor it locally to change the portion of the design affected by the new feature
- As a result you will always be improving the quality of your design, thereby making it easier to work with in the future

Test-Driven Design (TDD) - 2



Agile SDLC


Start work on release N+1
Concept (Iteration -1)
Select the Project
Inception (Iteration 0, Warm Up)
Initiate the Project
Construction Iterations
Deliver a working system which meets the changing needs of stakeholders
Transition (Release, End Game)
Deploy Release N into Production
Production
Operate and Support Release N
Retirement
Remove the system completely from production
- Identify potential projects
- Prioritize potential projects
- Develop initial vision
- Consider project feasibility
- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements envisioning
- Initial architecture envisioning
- Setup environment
- Active stakeholder participation
- Collaborative development
- Model storming
- Test driven design (TDD)
- Confirmatory testing
- Evolve documentation
- Internally deploy software
- Active stakeholder participation
- Final system testing
- Final acceptance testing
- Finalize documentation
- Pilot test the release
- Train end users
- Train production staff
- Deploy system into production
- Operate the system
- Support the system
- Identify defects and enhancements
- Remove the final version of the system-Data conversion
- Migrate users
- Update enterprise models

During iterations