CITS5502: Assignment 1 – Characteristics of a Software Process

Ash Tyndall, 20915779

August 17, 2014

Contents

1 Essay & Conclusions

Software is everywhere. In the modern world, nearly everything is controlled by, managed by or was ultimately created by software. Modern software carries high expectations, thus it is in the interest of organisations who develop software to create and enforce processes that ensure that software is be developed free of bugs, to requirements, within a sensible timeframe.

However, software engineering is also a notorious industry to develop processes for: Due to the breakneck pace at which client demands, available tools and technological norms evolve, there are multiple competing views on the best way to ensure the goal of quality software released inexpensively and rapidly is achieved.

A whole industry has sprung up around software development methodologies; processes that guide a team through the process of developing software in a way that hopes to achieve these goals. However, because this methodology industry in part is a for-profit one, the effectiveness or novelty of a given new process can take the backseat to marketing and economic concerns.

This leads us to ask the question; "how similar are different software processes?" When a new process appears, it can be useful to frame it in terms of differences to existing processes, and if the process has no significant differences, it can be helpful to know this easily. Finding these differences, as it turns out, is no simple task: A cursory internet search for each of these processes will yield a diagram of some form explaining how the process operates, however frequently these diagrams invent their own notation, which serves to obscure the process from comparison with other processes.

It is clear that if one wishes to compare these processes, one must first design a notation that can describe the core of their processes in a similar way. This was the first step in my task, which is discussed in section 2. My notation is relatively simple, but I believe it presents a good balance between complexity and an ability to represent the defining characteristics of different processes.

A key part of my notation is its brevity, which is partly enabled by the lack of defined decision nodes; a decision is instead represented by multiple paths leaving a given node on the graph, with labels on some or all of these exit paths. When there is an unlabelled path and other labelled paths, it is assumed that you will always follow the labelled path if you meet the criteria of the label, otherwise you will take the unlabelled path.

Another important part of my notation is the broader phase marker. This dashed box surrounds different sections of a process to indicate either that this section of nodes is considered a broader stage of the process or that there is some form of loop between the nodes represented in the section. This is not strictly necessary, but is provided for visual clarity.

Finally, a part of my notation that is seen in some processes is the presence of a "deliverable/artifact node" that does not trace its way back to the "Start" node of the process. This is seen for instance with the "Scrum" model in subsection 2.7. This situation indicates that the given process has a prerequisite that is expected will be fulfilled outside of the process scope. As different software processes have different scopes, it is useful in this way to represent how "out of scope" information interacts with the process.

Once these notations were completed for the ten existing software processes and my personal process, it was clear to see that the level of specification each process provides varies wildly, and seems to be only loosely correlated with its relationship to similar processes. For instance both Dynamic Systems Development Method (DSDM; subsection 2.6) and Scrum are both considered to be "Agile" processes, however the level of expression between their processes differs massively.

This example highlights a broader understanding I achieved regarding processes; there is a clear spectrum of formality in processes that appears to transcend the "genre" of the process. While the Agile processes I investigated here generally lent towards the "no formality" end of the spectrum, DSDM demonstrated that it is not necessary for this to be the case. While DSDM has codified several more steps than Scrum, it still ensures that there are heavy amounts of iteration in the process stages, a core principle of Agile.

The most linear of the processes I described was that of the Team Software Process (subsection 2.9), which appears to replace the concept of iteration with the presence of intense and numerous testing periods and post-mortems at each stage of the process. The most simple of the processes I described, in my opinion, was that of Scrum, which only had eight nodes, and a very simple iterative cycle of develop, meet, determine amount of development remaining, which was iterated over within a strict time-limit.

This notation exercise also highlighted the similarities between many processes that I chose to represent in my notation. For instance, broad similarly was found between my own personal process, and that of the Iterative Development Model, Rational Unified Process and Agile Unified Process. The differences between them and my own were primarily in that I lack a design prototype stage, and scope investigation is not emphasised in my personal process.

The similarity between those processes and my own was also emphasised with the creation of my taxonomy for software processes (section 3.) I elected to create a decision tree style system, whereby questions are asked and answered yes or no, until only one process matches the set of answers provided. The development of this tree was an interesting exercise in of itself, as while the questions could easily split the processes into a variety of different trees, it was important to find the right combination of questions to maximally represent the differences between the processes on the tree in the minimum amount of questions. This required considering a variety of combinations, until I settled on the set that can be seen in the aforementioned section.

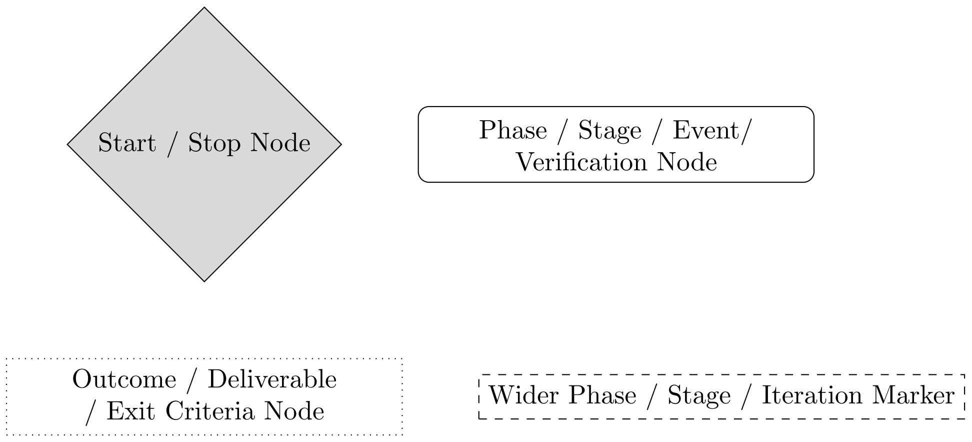
The taxonomy also exemplified some of the primary defining differences between the processes that I chose. The two primary questions were "Are requirements reevaluated throughout?" and "Is client feedback or acceptance part of evaluation?" The answers to these questions split the processes into a group of five and two groups of 3, which clearly demonstrated the different approaches to requirements and feedback within the processes I investigated.

In conclusion, we can see that many software processes that hold themselves as being different may not necessarily be as innovative as previously thought. While it is an involved task to take the varying source materials describing these processes and convert them into a common notation, there are large benefits to doing so, as they allow us to more easily see how the processes relate, their relative complexities, and how encompassing the iterative aspects of these processes are.

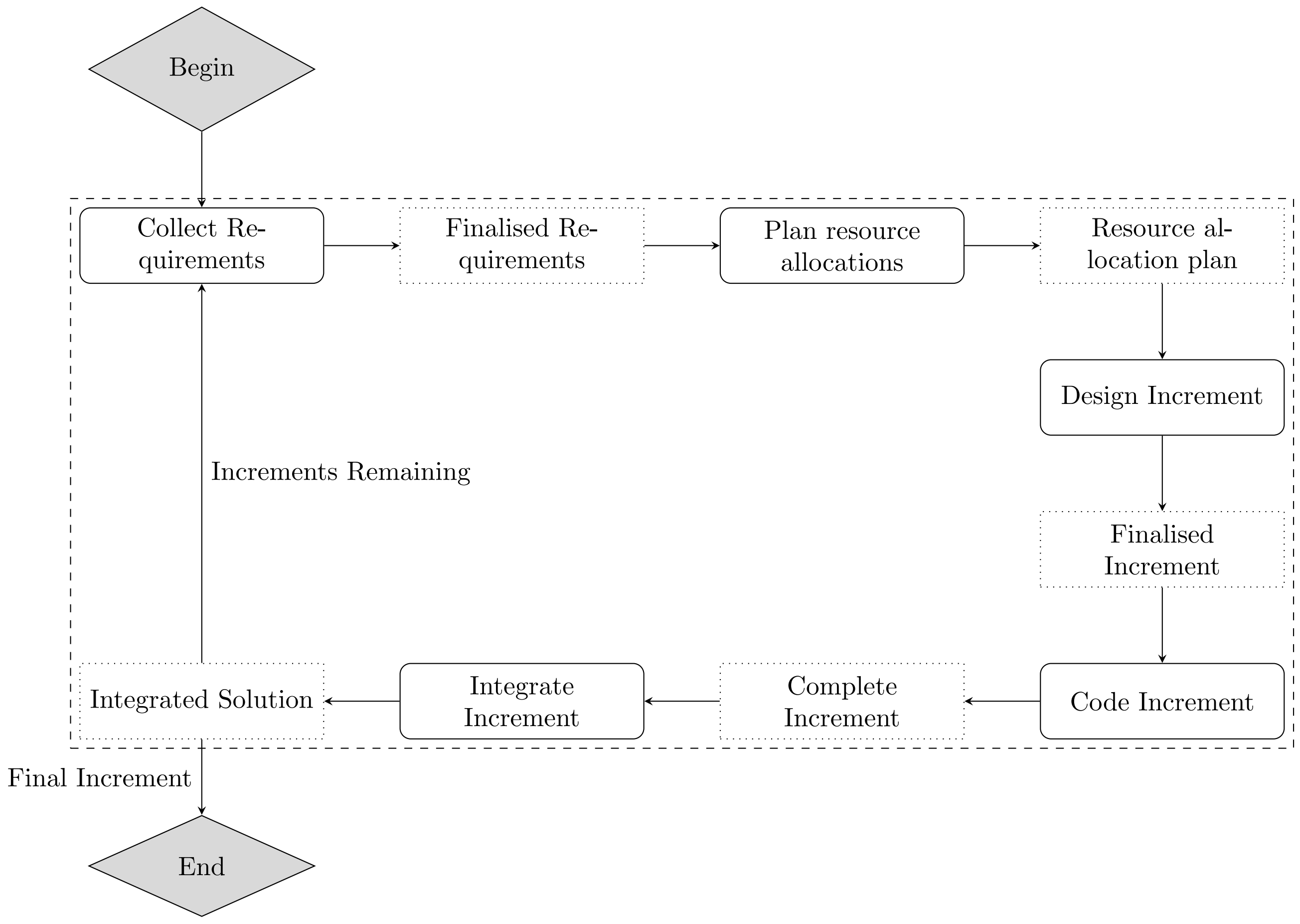
The taxonomy created also assisted in defining the differences between processes, and could be a useful tool in future, as it can be indefinitely extended with new questions to differentiate between new processes. Together, this notation and taxonomy form a useful set of tools to critically evaluate existing and new software processes, to determine their true innovation, and if that innovation is different enough to justify the cost of embracing them.

2 Processes Notation

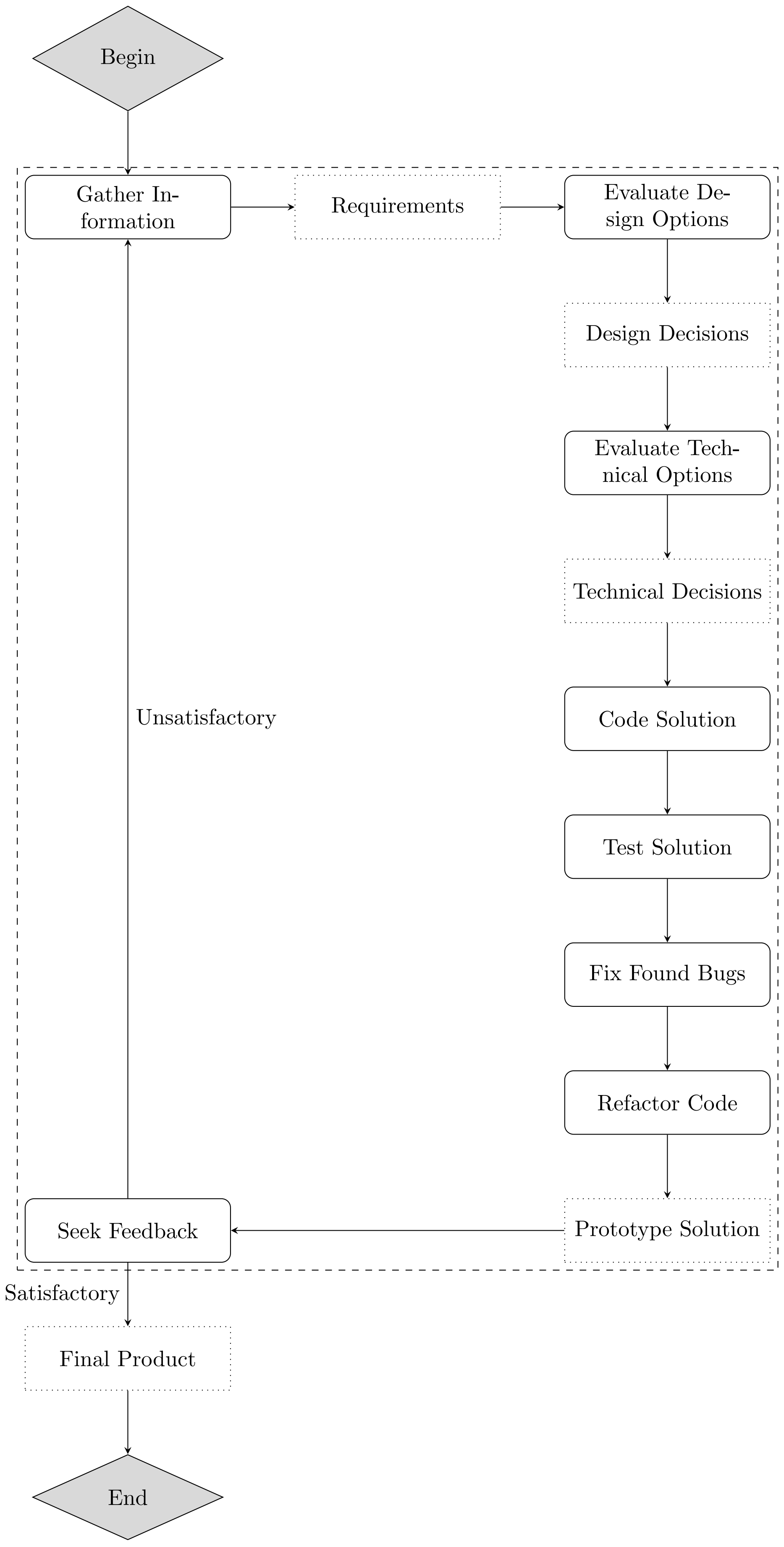
2.1 Key


Start / Stop Node
Phase / Stage / Event/ Verification Node
Outcome / Deliverable / Exit Criteria Node
Wider Phase / Stage / Iteration Marker

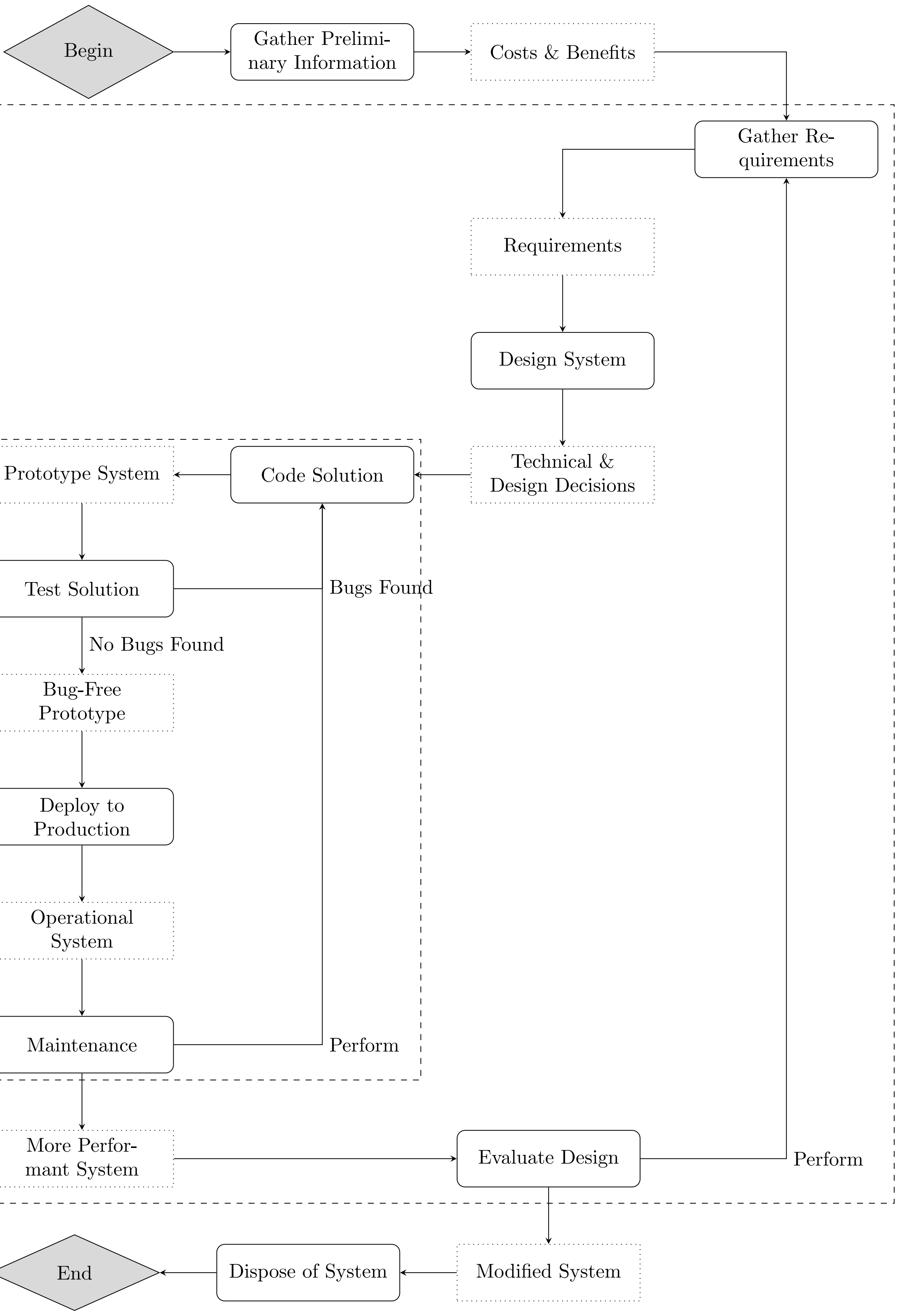
2.2 Incremental Build Model



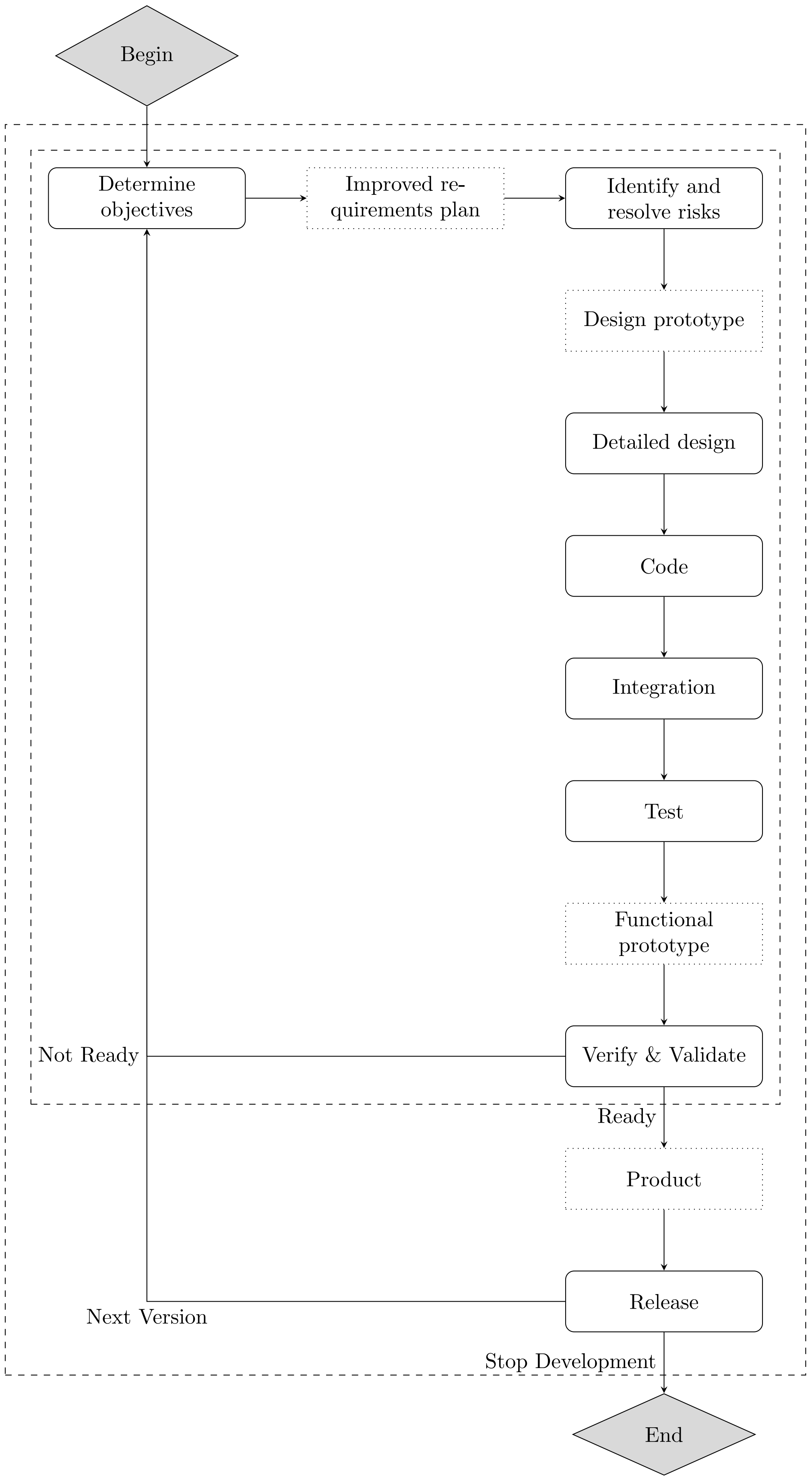
2.3 Personal Process



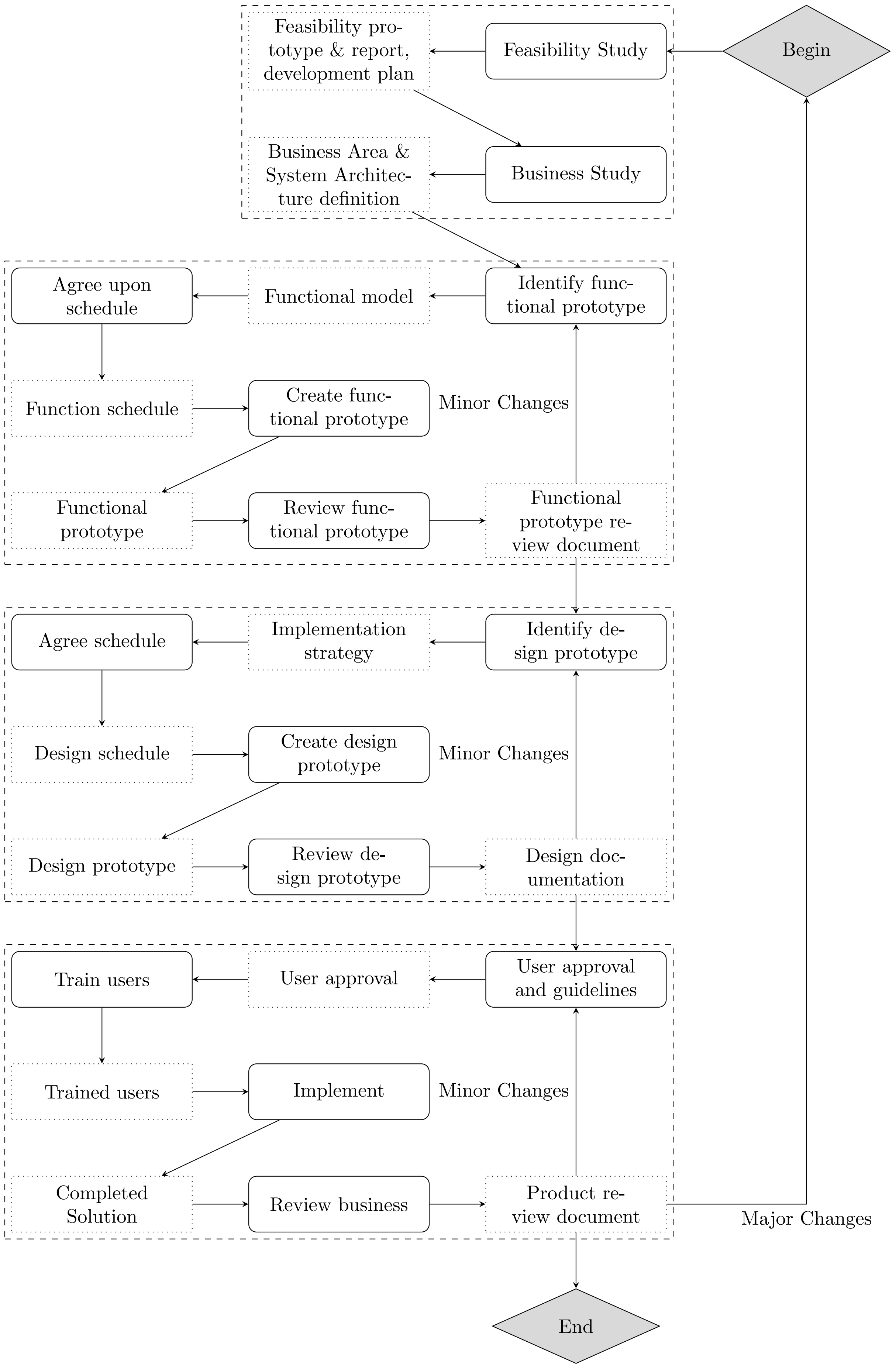
2.4 Systems Development Life Cycle



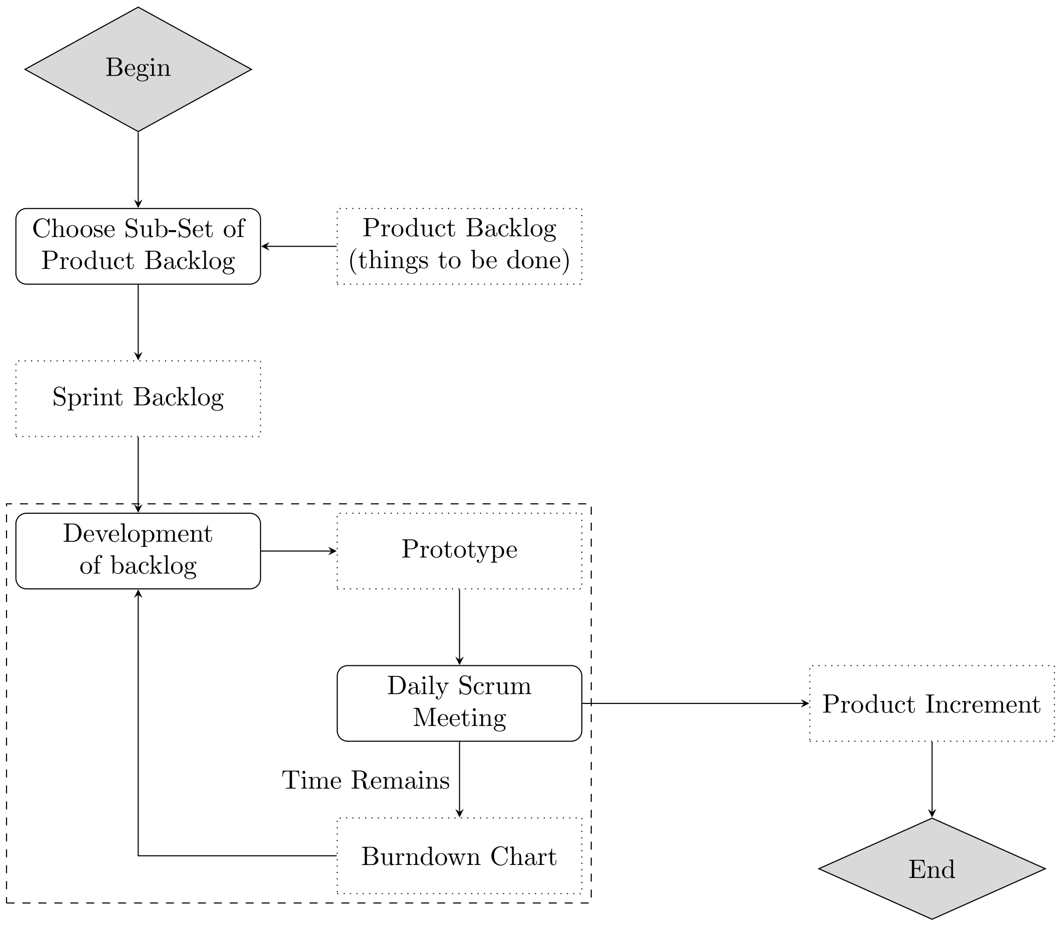
2.5 Spiral Model



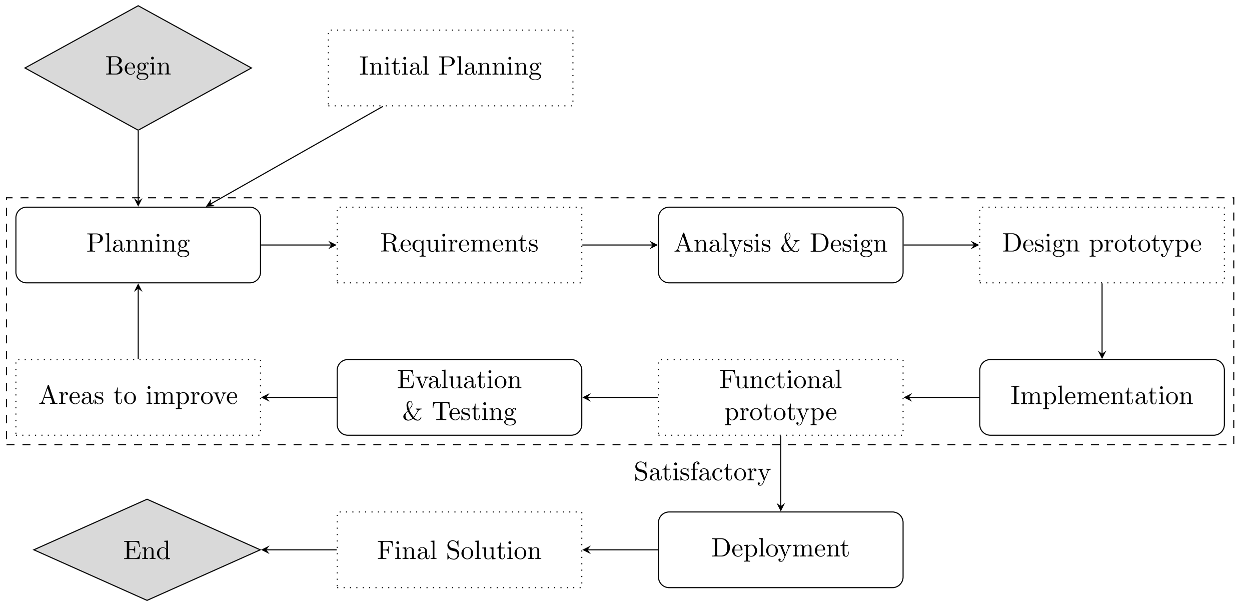
2.6 Agile: Dynamic Systems Development Method



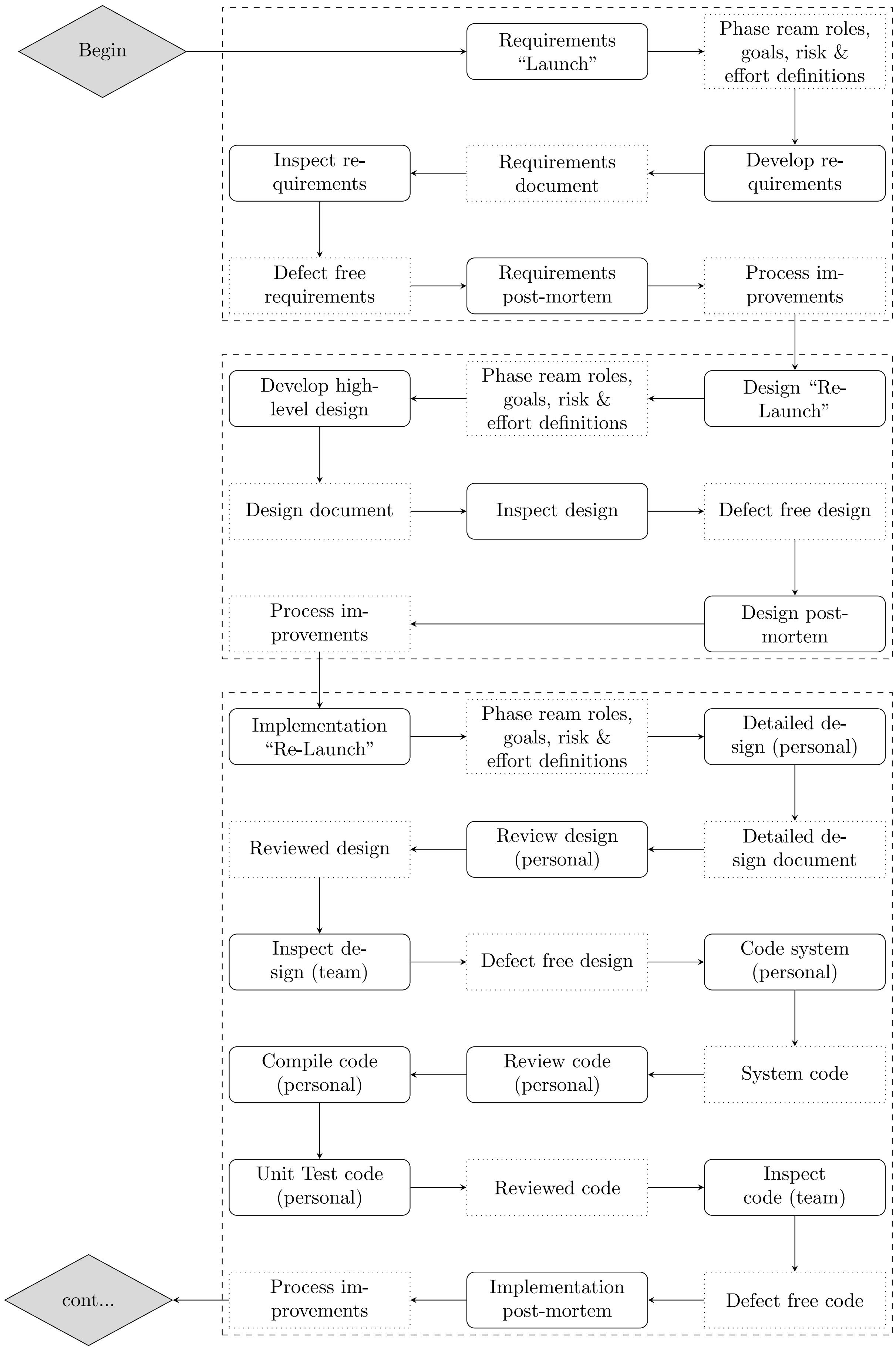
2.7 Agile: Scrum

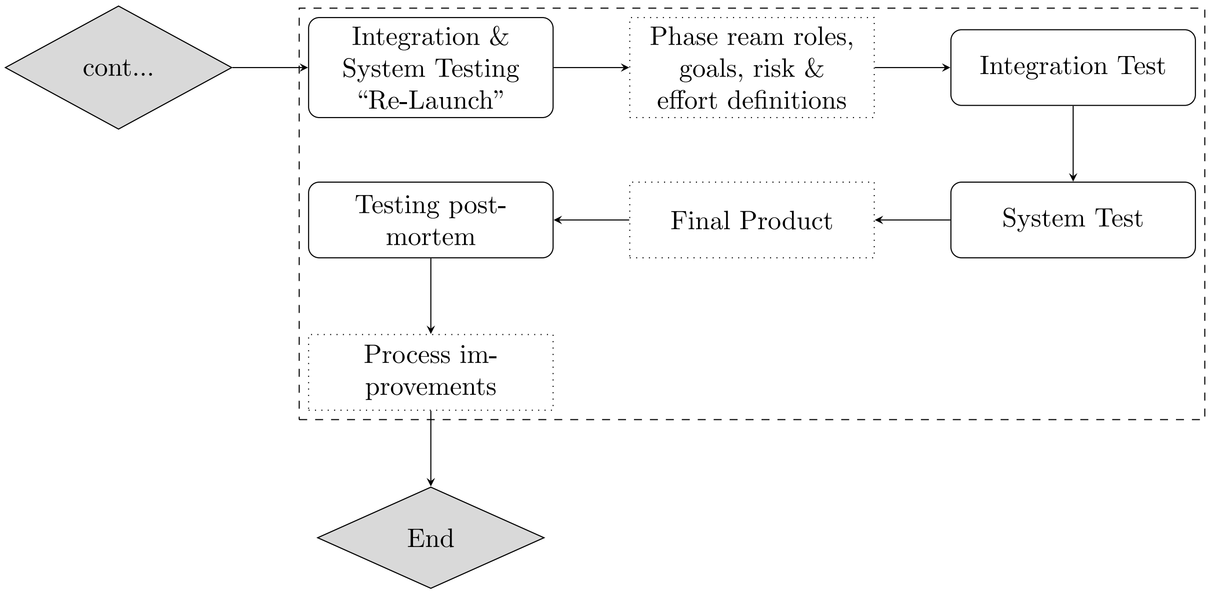


2.8 Iterative Development Model

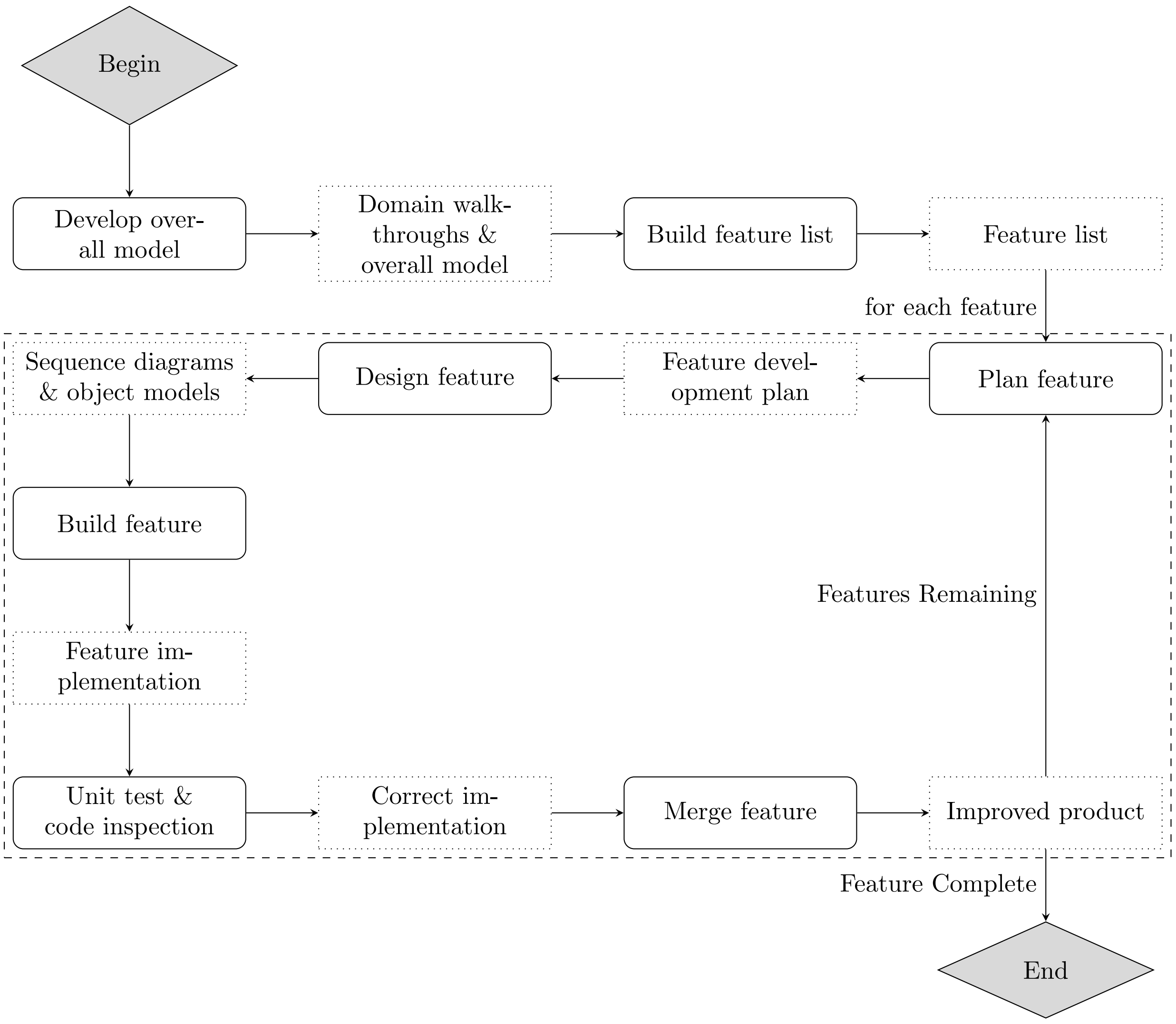


2.9 Team Software Process

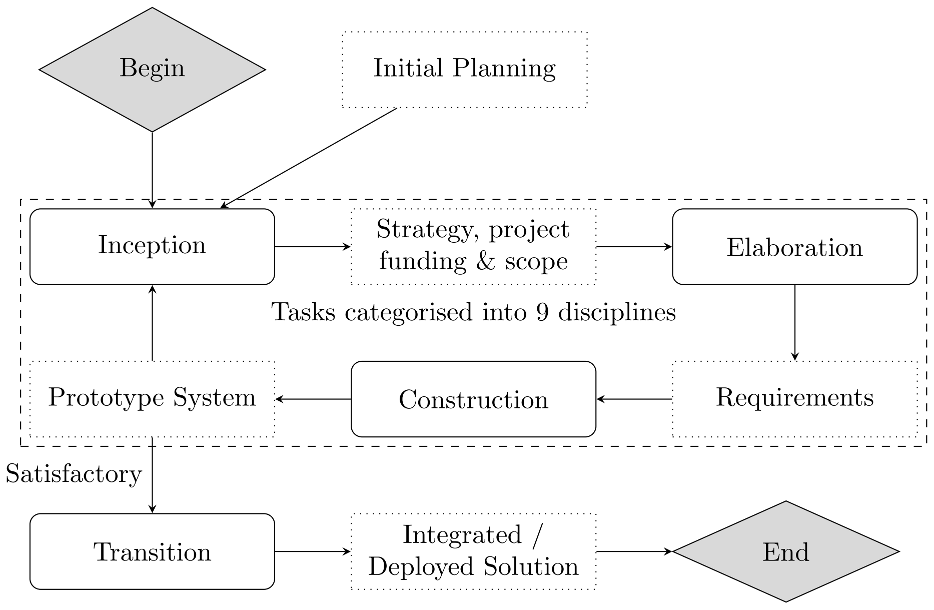



cont...
Integration & System Testing "Re-Launch"
Phase ream roles, goals, risk & effort definitions
Integration Test
System Test
Final Product
Testing post-mortem
Process improvements
End

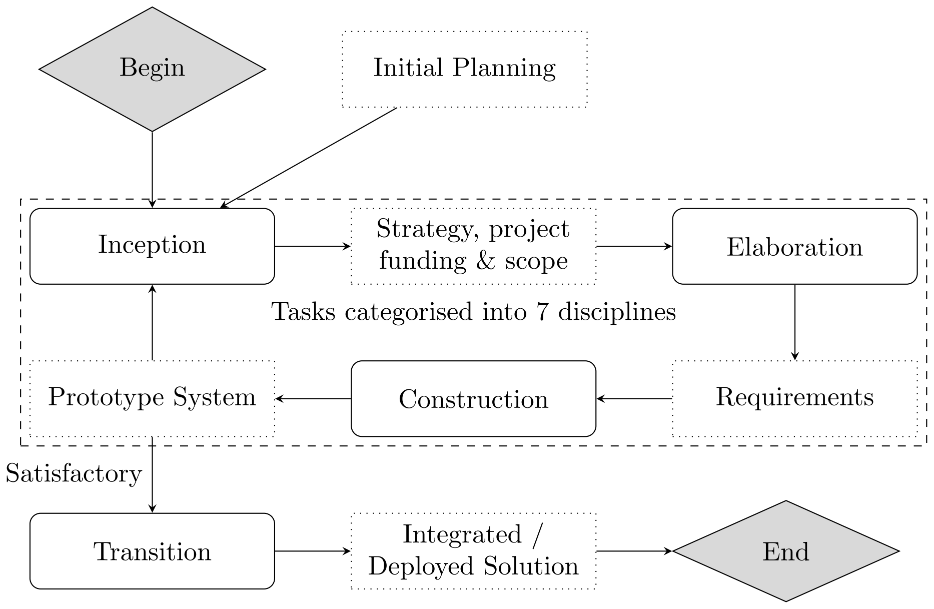
2.10 Feature Driven Development



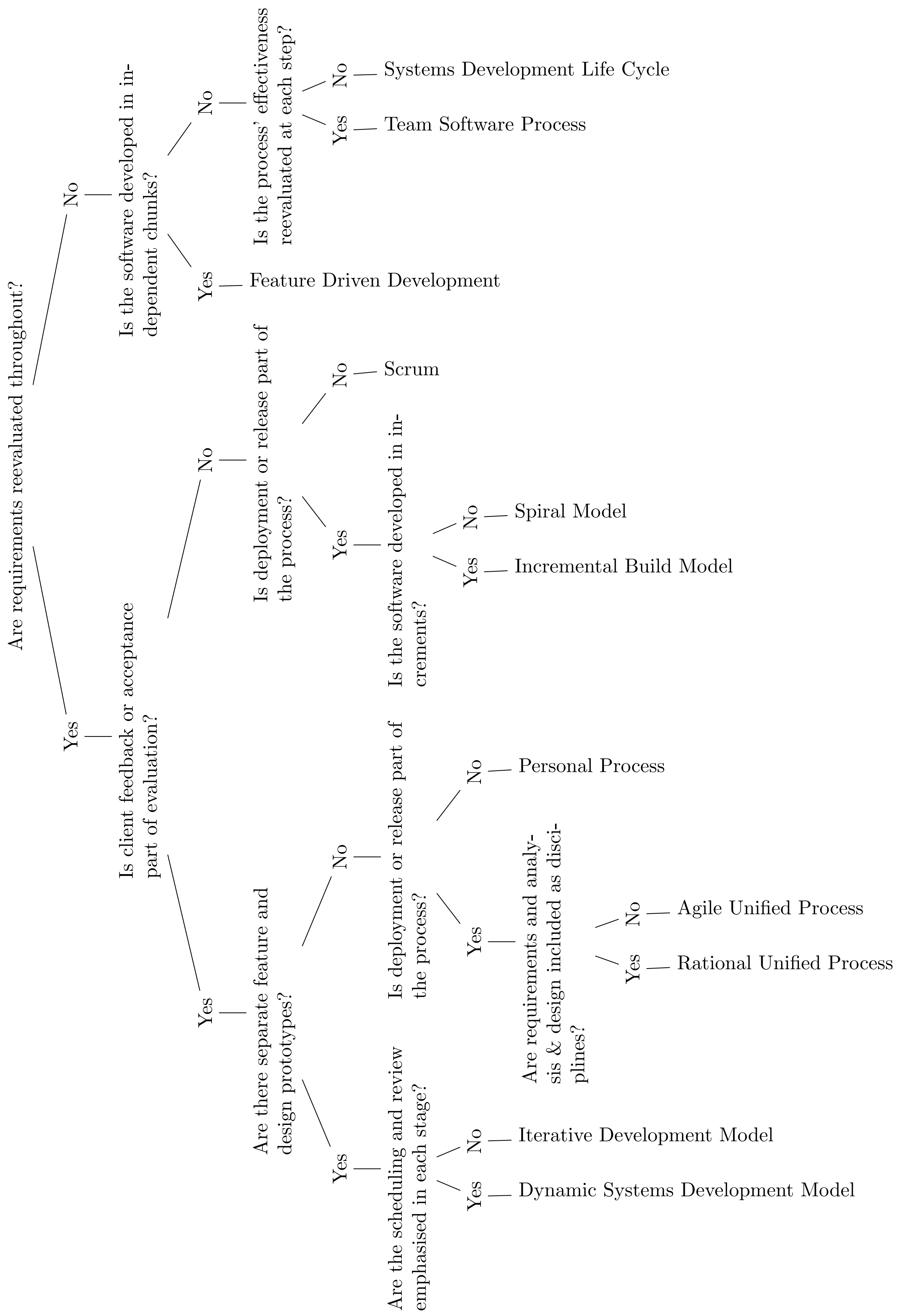
2.11 Rational Unified Process



2.12 Agile Unified Process



3 Taxonomy



Are requirements reevaluated throughout?
No
Is the software developed in independent chunks?
No
Is the process' effectiveness reevaluated at each step?
No
Systems Development Life Cycle
Yes
Team Software Process
Yes
Feature Driven Development
Yes
Is client feedback or acceptance part of evaluation?
No
Is deployment or release part of the process?
No
Scrum
Yes
Is the software developed in increments?
No
Spiral Model
Yes
Incremental Build Model
Yes
Are there separate feature and design prototypes?
No
Is deployment or release part of the process?
No
Personal Process
Yes
Are requirements and analysis & design included as disciplines?
No
Agile Unified Process
Yes
Rational Unified Process
Yes
Are the scheduling and review emphasised in each stage?
No
Iterative Development Model
Yes
Dynamic Systems Development Model

4 References

[1] Incremental Build Model, http://www.technotrice.com/incremental-model-in-software-engineering/, Accessed: August 17, 2014

[2] Systems Develop Life Cycle, http://www.computerworld.com/s/article/71151/System_Development_Life_Cycle, Accessed: August 17, 2014

[3] Spiral Model, http://commons.wikimedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg, Accessed: August 17, 2014

[4] Dynamic Systems Development Method, http://commons.wikimedia.org/wiki/File:Processdata-FMI.png, Accessed: August 17, 2014

[5] Scrum, http://www.mountaingoatsoftware.com/uploads/blog/ScrumMediumLabelled.png, Accessed: August 17, 2014

[6] Iterative Development Model, http://commons.wikimedia.org/wiki/File:Iterative_development_model.svg, Accessed: August 17, 2014

[7] Team Software Process, Page 33, http://cs.mty.itesm.mx/profesores/pverdines/TCS/p3/PSP-TSP.pdf, Accessed: August 17, 2014

[8] Feature Driven Development, http://commons.wikimedia.org/wiki/File:Fdd_process_diagram.png, Accessed: August 17, 2014

[9] Rational Unified Process, http://projects.staffs.ac.uk/suniwe/project/projectapproach.html, Accessed: August 17, 2014

[10] Agile Unified Process, http://www.ambysoft.com/unifiedprocess/agileUP.html, Accessed: August 17, 2014

Appendices

A Personal Process Program

A.1 Source Code

```
# Written in Python 3.3
from decimal import *
from math import radians
import itertools

# In degrees
PLACES = {
  'uwa':        ('-31.981179', '115.81991'),
  'perth':      ('-31.954265', '115.8523935'),
  'greenwich':  ('51.4825766', '-0.0076589'),
  'southpole':  ('-85', '0'),
}

RADIUS_EARTH = 6378


# Uses the Haversine formula
# Assumes spherical earth and radian coordinates
def distance(coord1, coord2):
  from math import asin, sqrt, cos, sin

  (lat1, long1), (lat2, long2) = coord1, coord2

  def haversin(v):
    return (sin(v/2))**2

  return 2 * RADIUS_EARTH * asin(
      sqrt(
        haversin(lat2 - lat1) +
        cos(lat1) * cos(lat2) * haversin(long2 - long1)
      )
    )


if __name__ == '__main__':
  # Convert string coords into Decimal radian objects
  conv_places = {
    x : ( radians(Decimal(y1)), radians(Decimal(y2)) )
    for x, (y1, y2) in PLACES.items()
  }

  print('Distance calculations!')
  print()

  # Iterate over all place combinations and print the answers
  for (n1, c1), (n2, c2) in itertools.combinations(conv_places.items(), 2):

    print('Distance from {} to {}'.format(n1, n2))
    print('  = {}'.format(distance(c1, c2)))
    print()
```

A.2 Output

```
Distance calculations!

Distance from perth to uwa
  = 4.287899604948252

Distance from perth to greenwich
  = 14485.978495782614

Distance from perth to southpole
  = 6715.917481245382

Distance from uwa to greenwich
  = 14485.767563601663

Distance from uwa to southpole
  = 6712.656749767587

Distance from greenwich to southpole
  = 15192.844590121906
```

A.3 Development Timings

Timings have been submitted to lecturer.