We need Software Engineering!

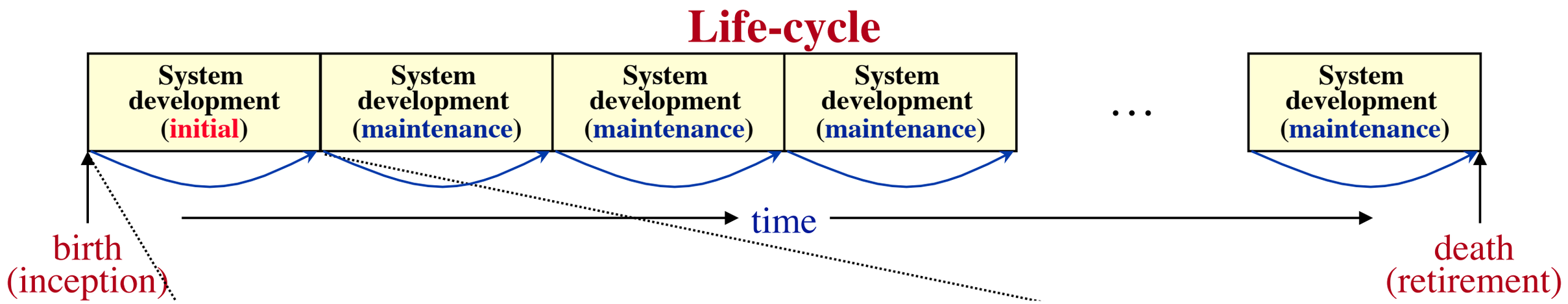
- Software is complex
- No solo-player please
- Software Engineering
  - Better design
  - Better management
  - Better process

Today: Software Process

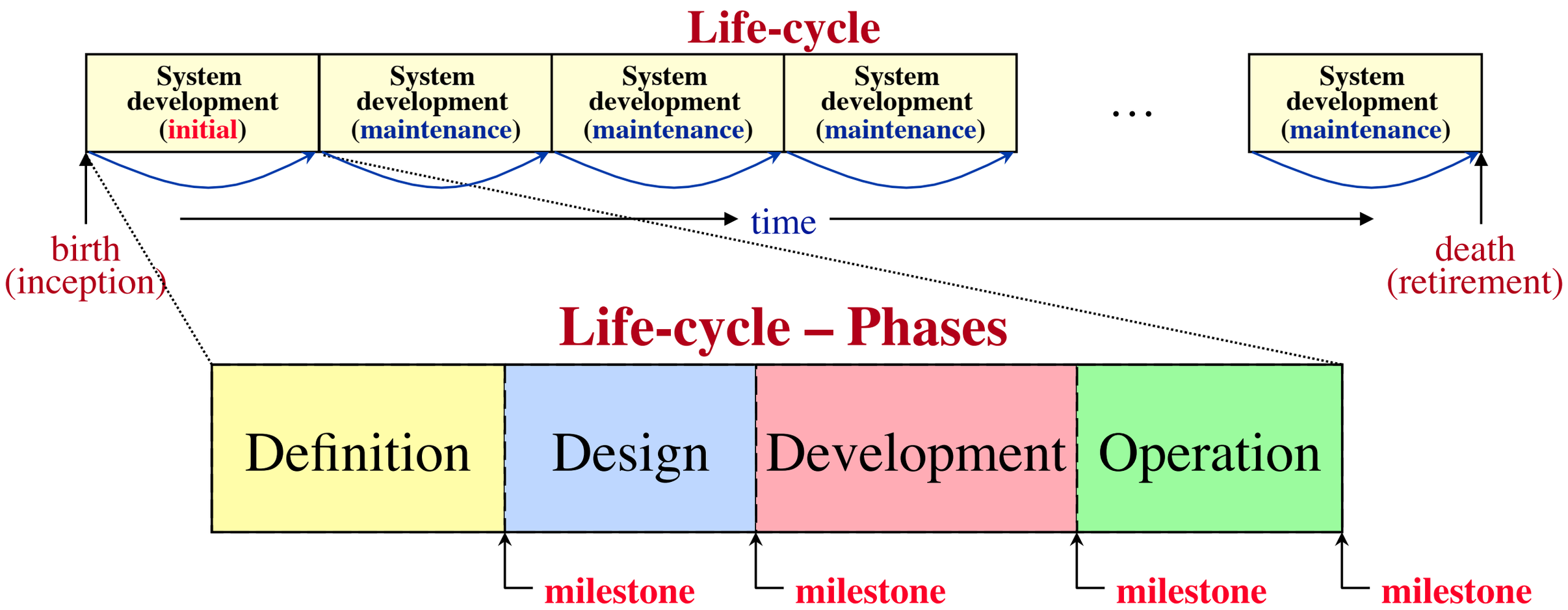
Types of software development projects

- Green field projects: new development
- **Evolutionary projects**: maintenance
  - corrective: fix defects
  - adaptive: adapt to new technology, new laws, etc.
  - perfective (refactoring): make more maintainable
  - enhancement: add new features
- Framework/component projects: reuse

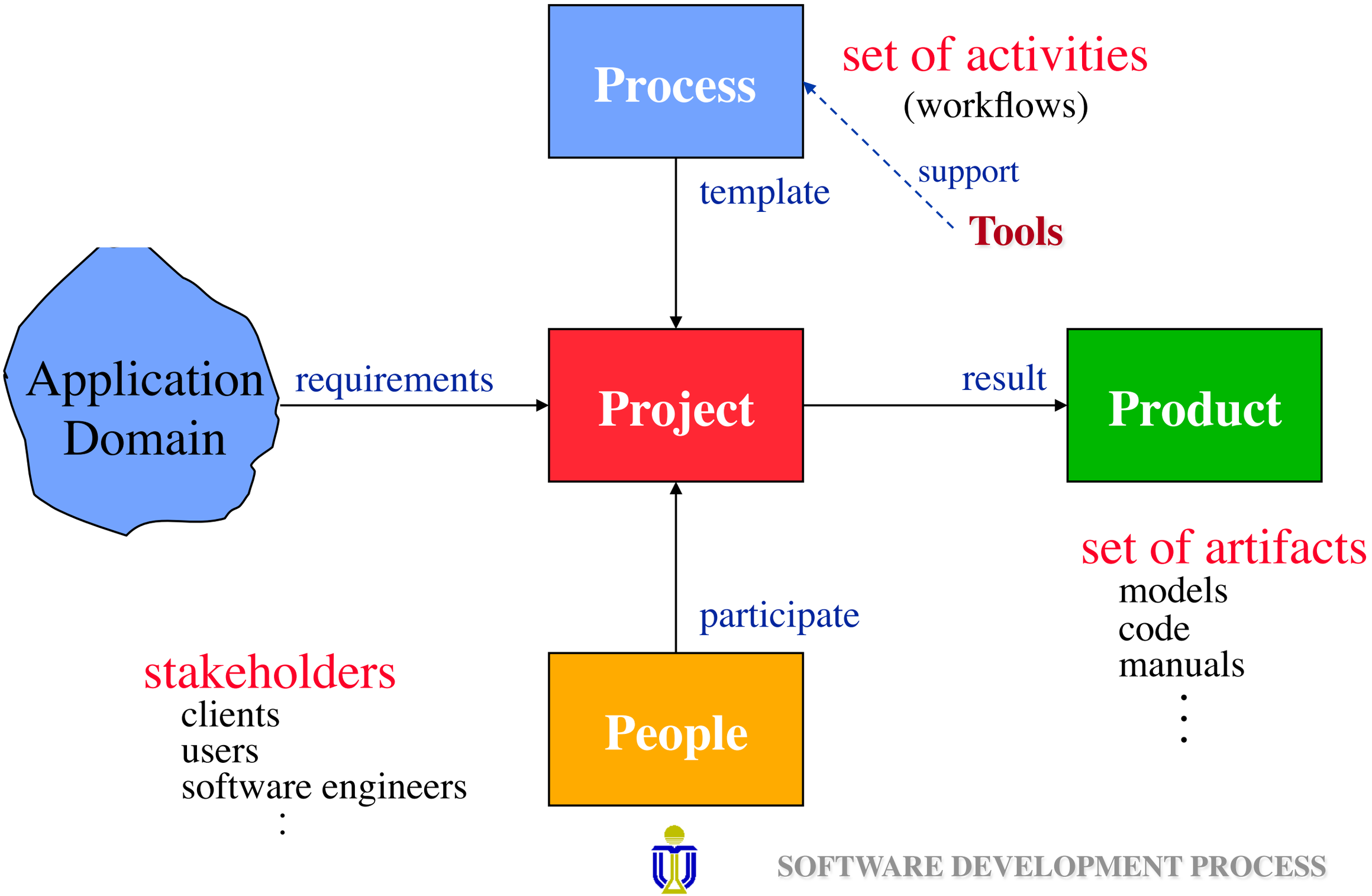
SOFTWARE DEVELOPMENT LIFE-CYCLE



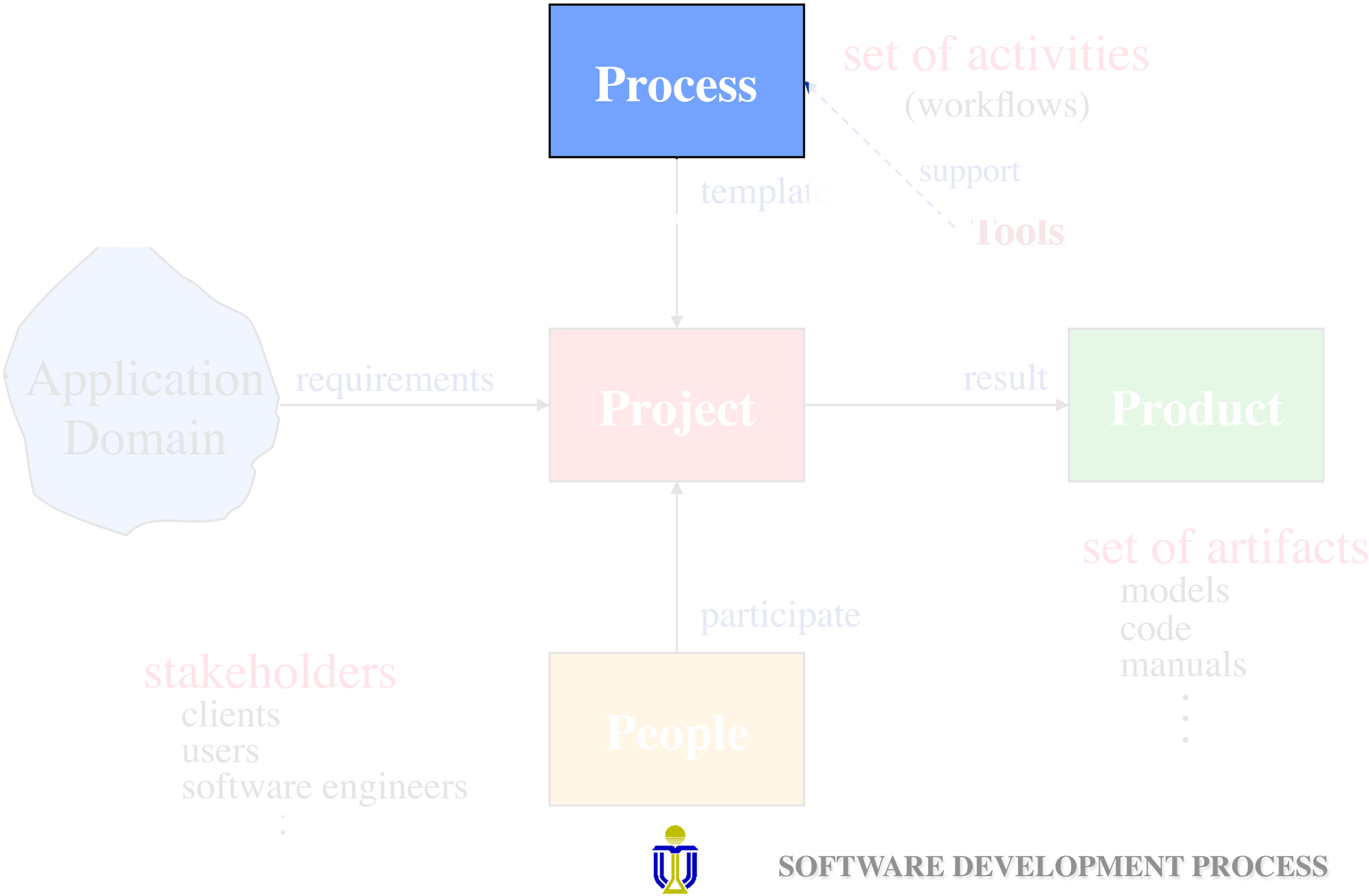
SOFTWARE DEVELOPMENT LIFE-CYCLE



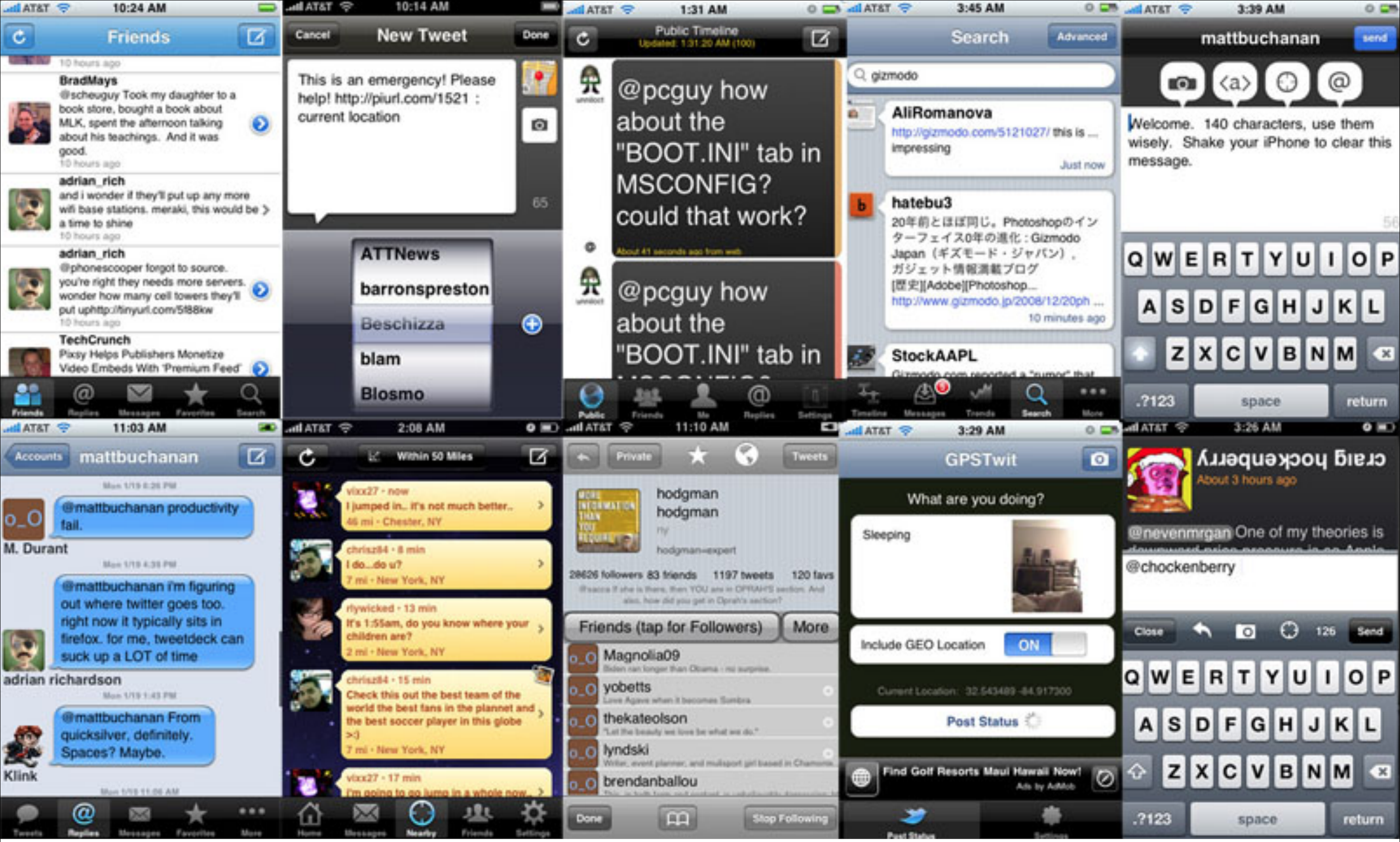
THE FOUR P'S IN SOFTWARE DEVELOPMENT



THE FOUR P'S IN SOFTWARE DEVELOPMENT

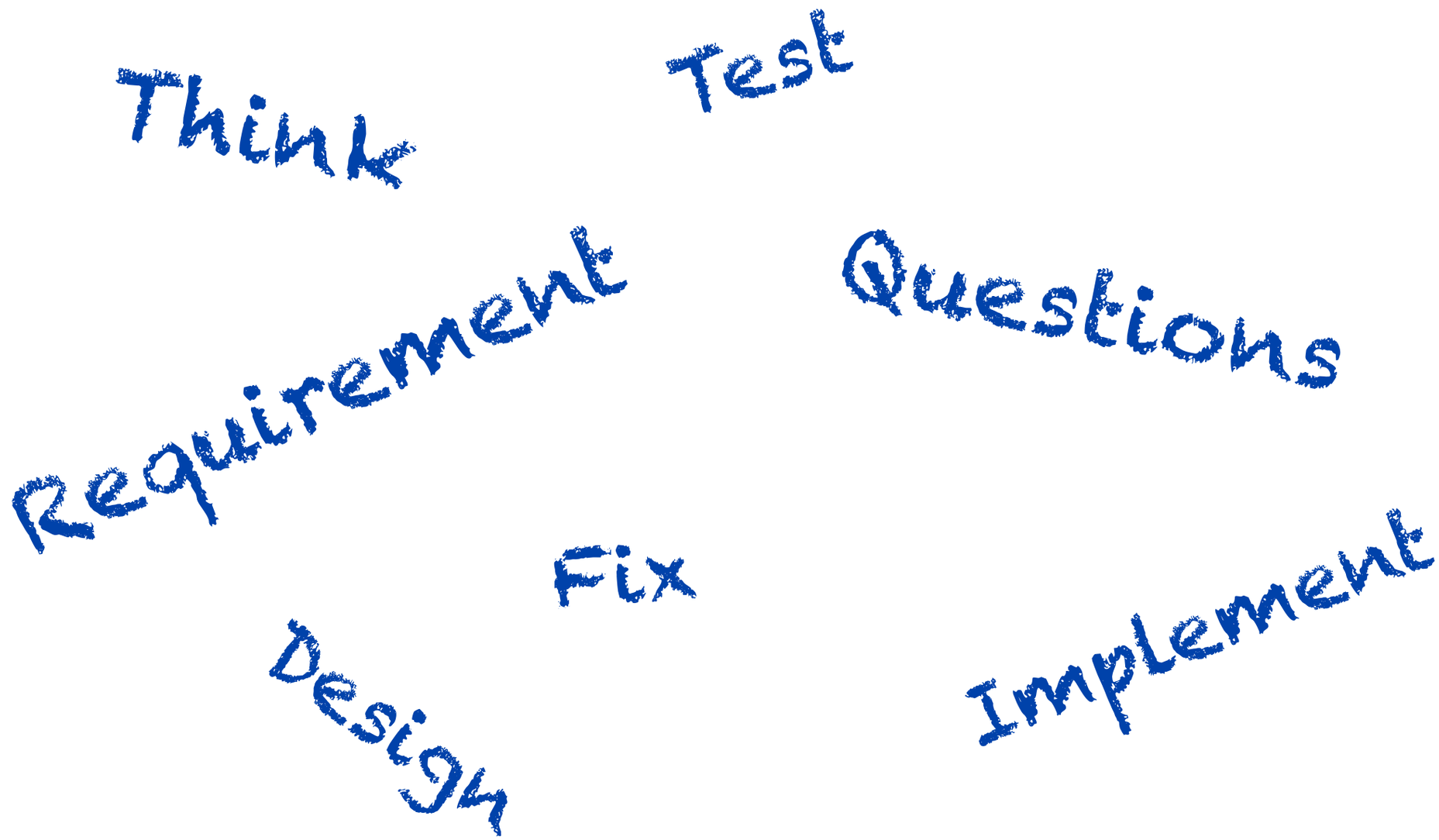



Let's make a nice program!
dipart.com
You



http://cache.gawkerassets.com/assets/images/gizmodo/2009/01/twitterappmain.jpg

Common stages



Development process stages

- Most software development processes share stages
  - **gathering** the system **requirements**
  - **analyzing** and **designing** the system
  - **implementing** the system
  - **testing** the system
  - **maintaining** the system

Development process stages

- They differ in how these stages are
  - combined
  - emphasized
  - carried out
- Need to understand what works best in practice (for you) and why

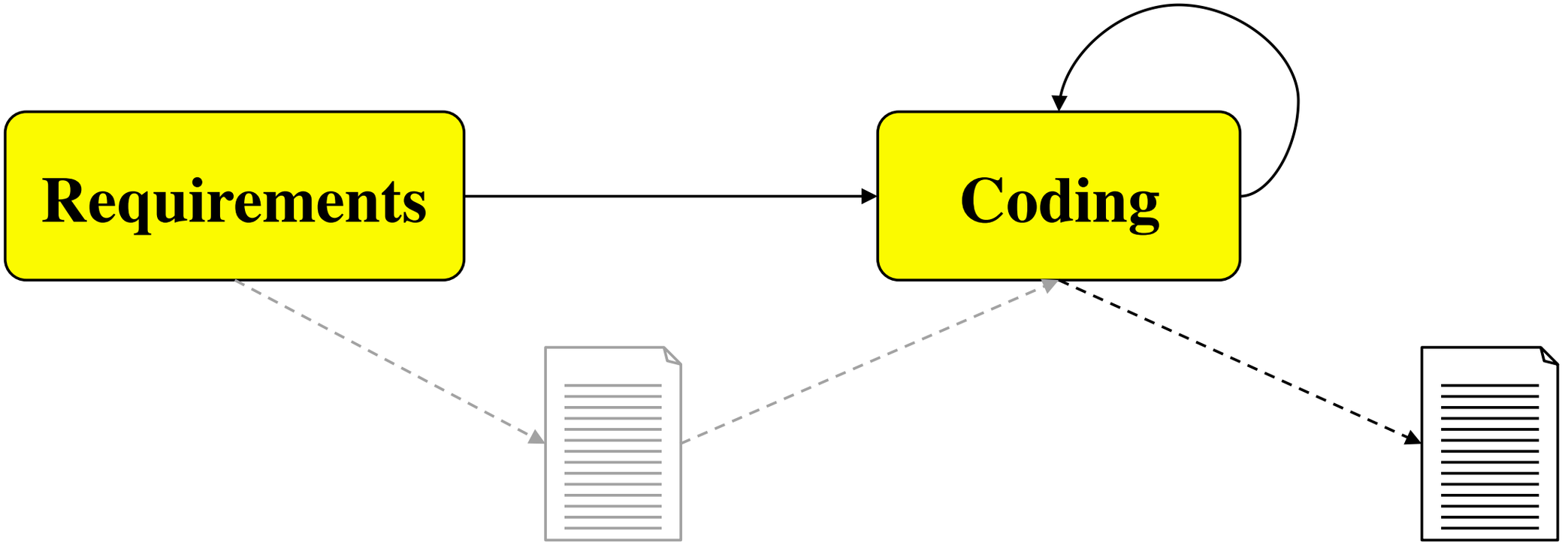
Software Development Processes

- Code and Fix
- Waterfall
- Prototyping
- Spiral
- Phased
- Agile
- Open source
- Task oriented
- UP

Points to keep in mind

- Understand
  - the strengths (pros) and
  - weaknesses (cons) of different development processes
- Different processes are not mutually exclusive
  - Often they are used together in practice

CODE-AND-FIX SOFTWARE DEVELOPMENT PROCESS



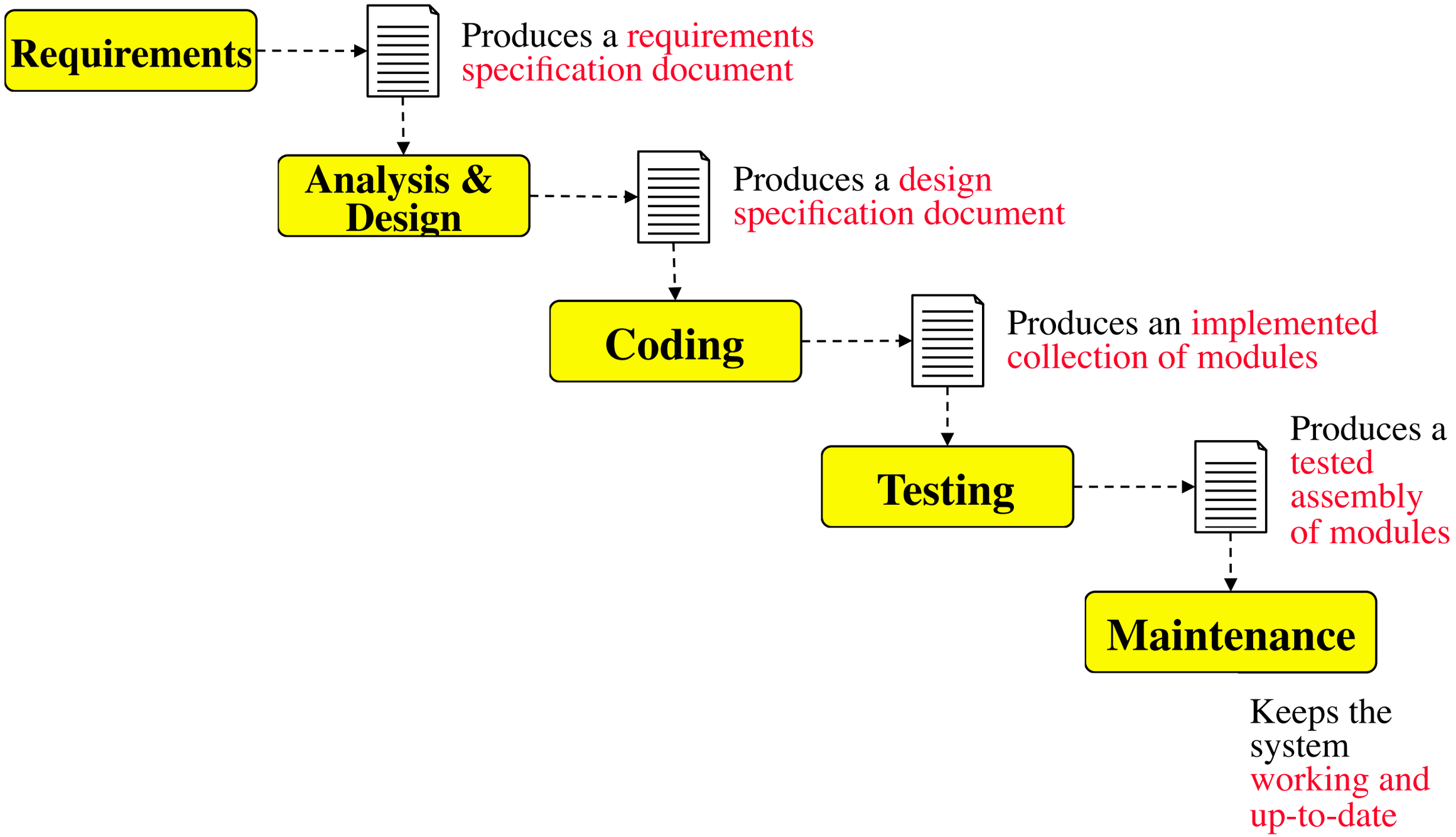
Code-and-Fix

- Many changes (without requirements/plans)
  - Code structure often becomes messy
- Unsuitable for large systems
  - turnover of personnel
  - difficult to understand and maintain code
  - requirements can easily be unmatched
- The software development process becomes
  - unpredictable and uncontrollable
  - over schedule, over budget and low quality

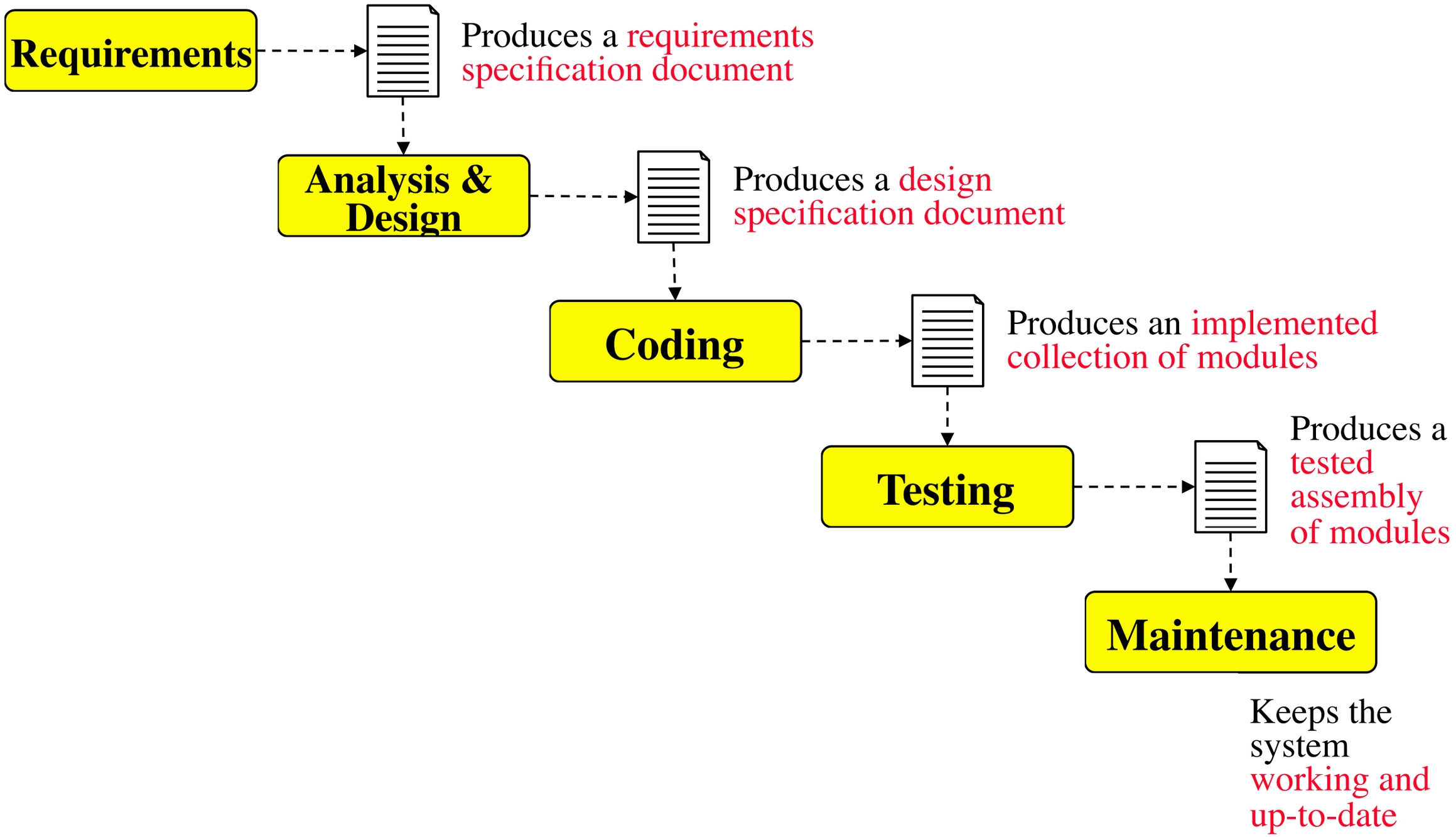
Code-and-Fix

- Many changes (without requirements/plans)
  - Code structure often becomes messy
- Unsuitable for large systems
  - turnover of personnel
  - difficult to understand and maintain code
  - requirements can easily be unmatched
- The software development process becomes
  - unpredictable and uncontrollable
  - over schedule, over budget and low quality

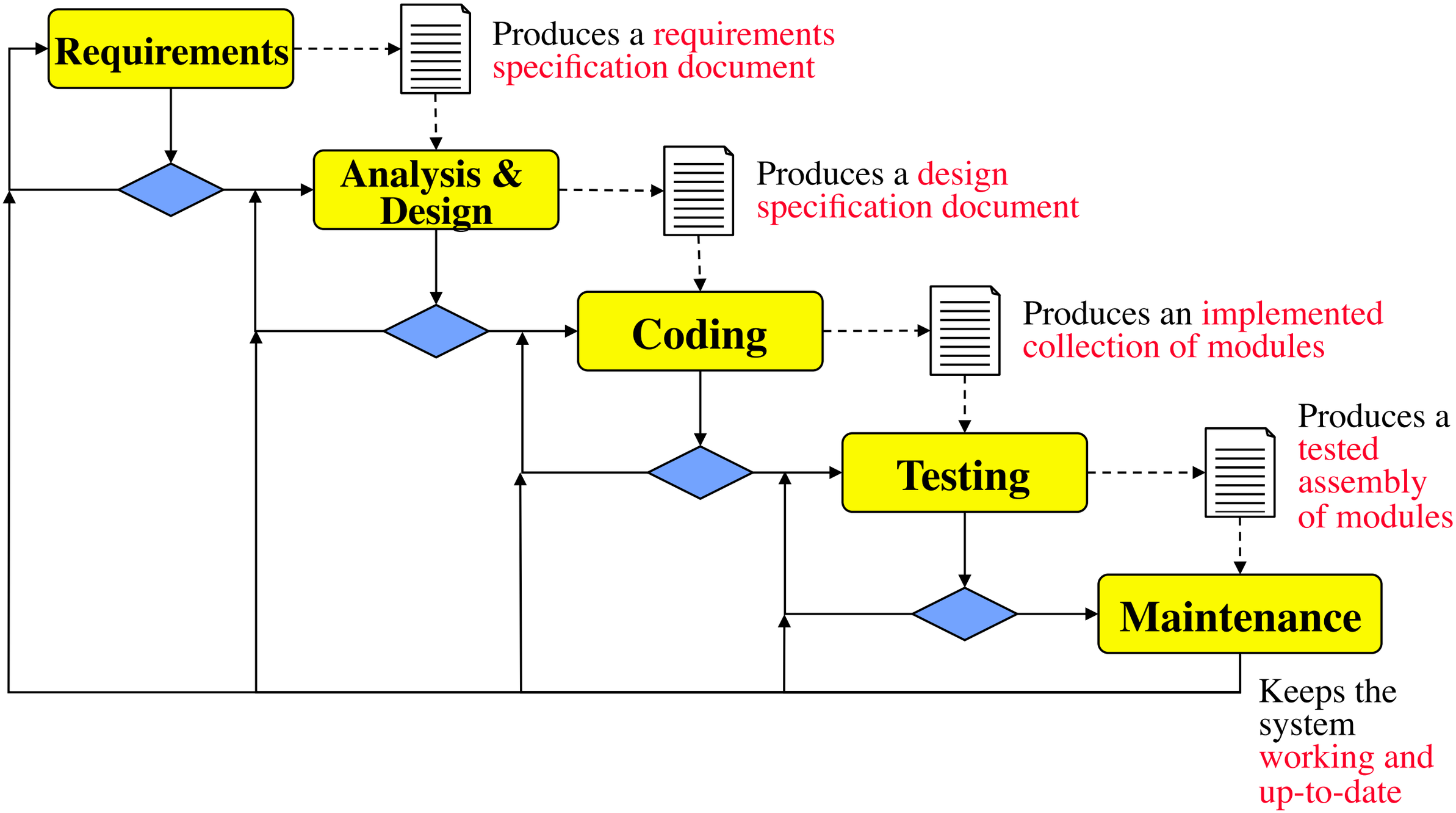
WATERFALL SOFTWARE DEVELOPMENT PROCESS



WATERFALL SOFTWARE DEVELOPMENT PROCESS



WATERFALL SOFTWARE DEVELOPMENT PROCESS



Waterfall: Pros

- Imposes needed discipline (rigor and formality)
  - a systematic, sequential and phased approach
- Keeps the development process predictable and easy to monitor
- Enforces standards
  - requiring production of certain documents
  - requiring approval before proceeding
- Fits well with other engineering process models
  - (e.g., for hardware)

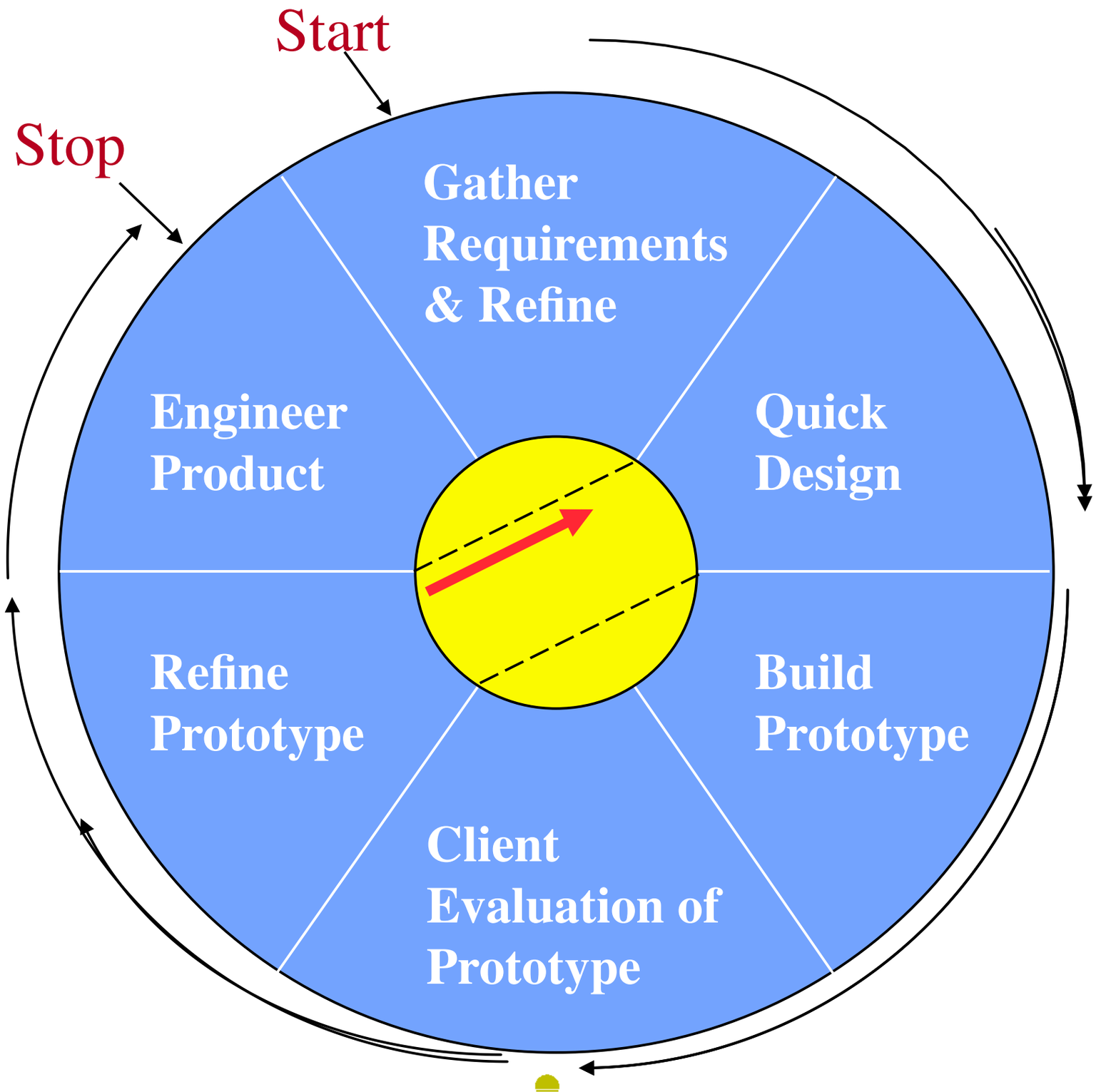
Waterfall: Cons

- Assumes a linear development
  - sequential flow of phases
- Assumes results of each phase can be frozen before proceeding to the next phase
  - Revision of requirements is difficult
- All planning is oriented toward a single product delivery date
  - The product usually is not available for feedback until the end of the development (high risk)

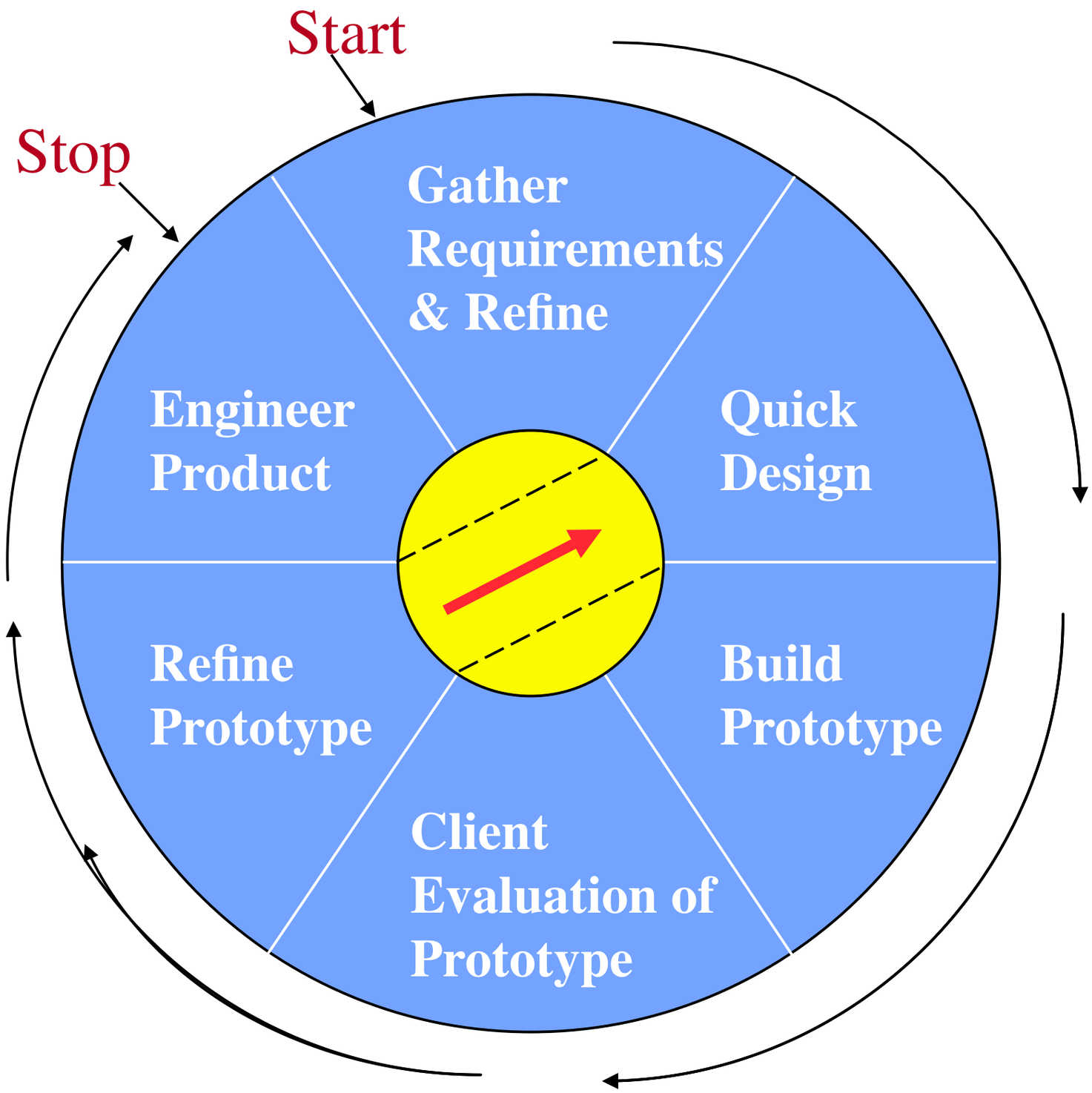
PROTOTYPING SOFTWARE DEVELOPMENT PROCESS

http://semanticstudios.com/publications/semantics/000228.php

PROTOTYPING SOFTWARE DEVELOPMENT PROCESS



PROTOTYPING SOFTWARE DEVELOPMENT PROCESS



- Basically a code-and-fix process **BUT** ...
  - enforces discipline
  - produces all required artifacts
- It is useful when requirements are vague or unknown as it allows exploration of
  - user interface
  - functionality needed

What to do with the final prototype?

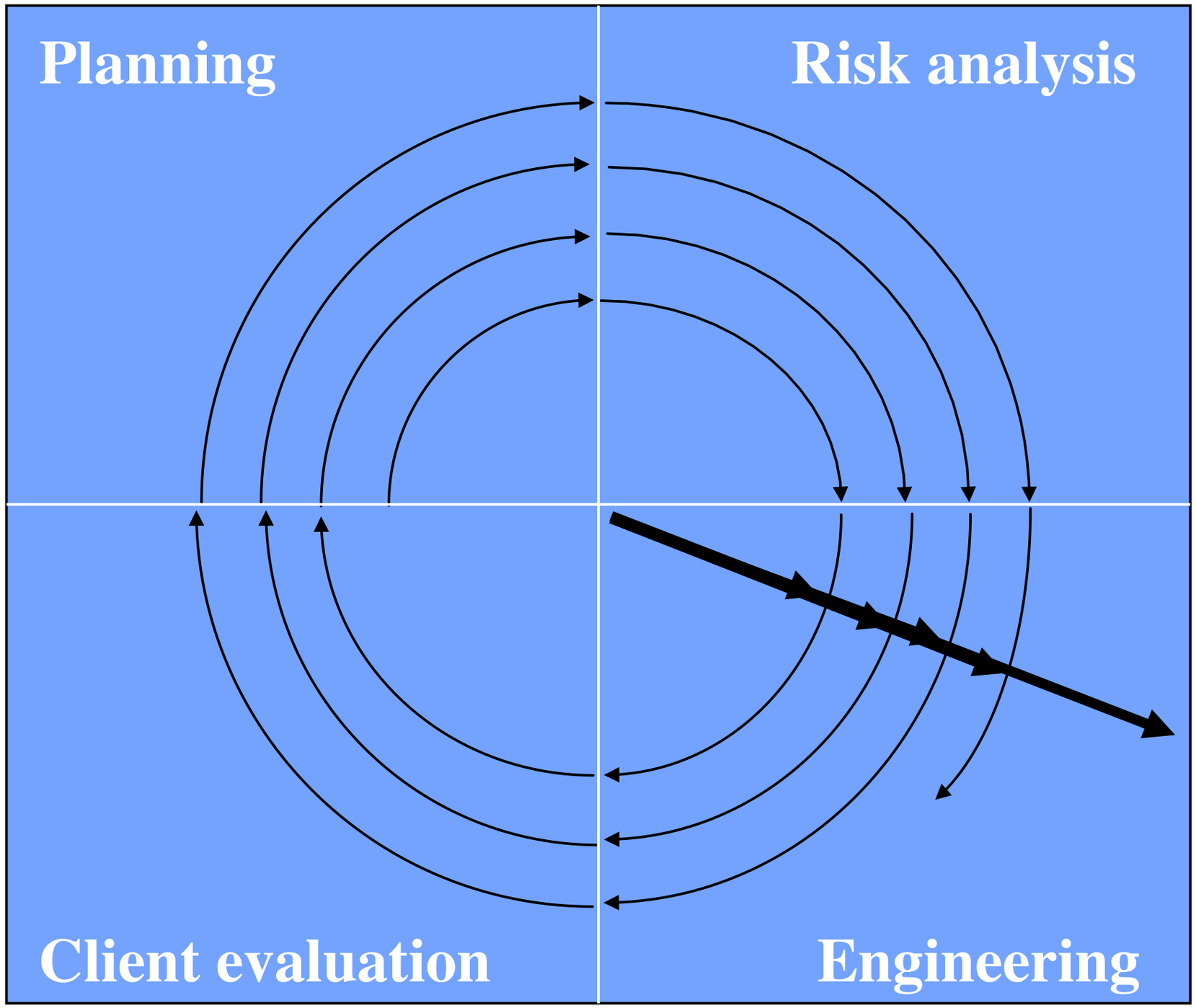
Prototyping: Pros

- Allows requirements to be quickly explored
  - Get a clear understanding of the system functionality
- Allows user feedback and approval to be obtained
- Allows different solutions to be explored
  - Have a higher chance of finding the “right” solution

Prototyping: Cons

- Not really a complete software development process
- The final "product" is not a complete system
- We usually throw out the system and rebuild

SPIRAL SOFTWARE DEVELOPMENT PROCESS



Go, no-go decision

Toward a completed system

RISK:

anything that can go wrong (endanger success)

- Technical risks
  - building the right system
  - system architecture
  - new technologies
  - performance
- Non-technical risks
  - right expertise
  - needed training
  - tight schedule
  - timely approvals

Dealing with Risks (ACMM)

- Avoid
  - re-plan or change requirements
- Confront
  - restrict the scope of its effect
- Mitigate
  - devise tests to see if it occurs
- Monitor
  - constantly be on the lookout for it

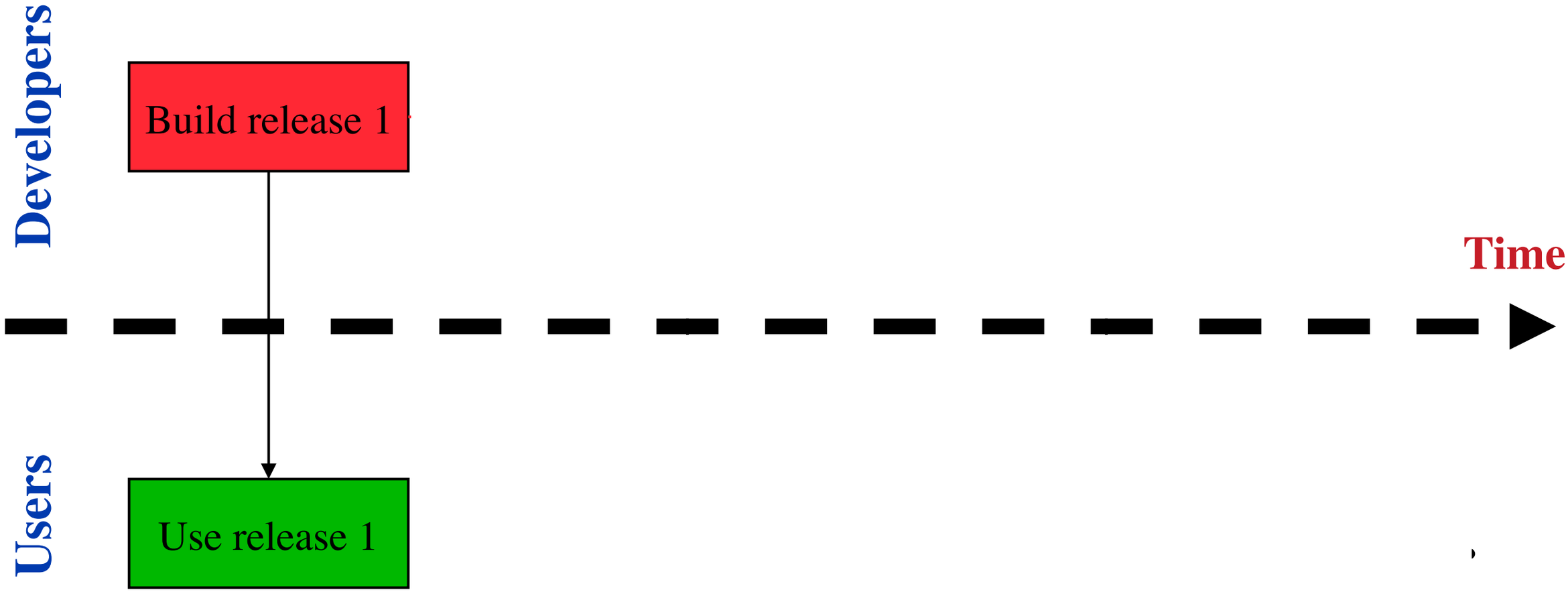
Spiral: Pros

- Accommodates good features of previous processes (Waterfall, prototyping) while risk-approach reduces their problems
- Focused early attention on options involving reuse
  - reduces risk
- Focuses on early eliminations of errors and weak solutions
  - reduces risk

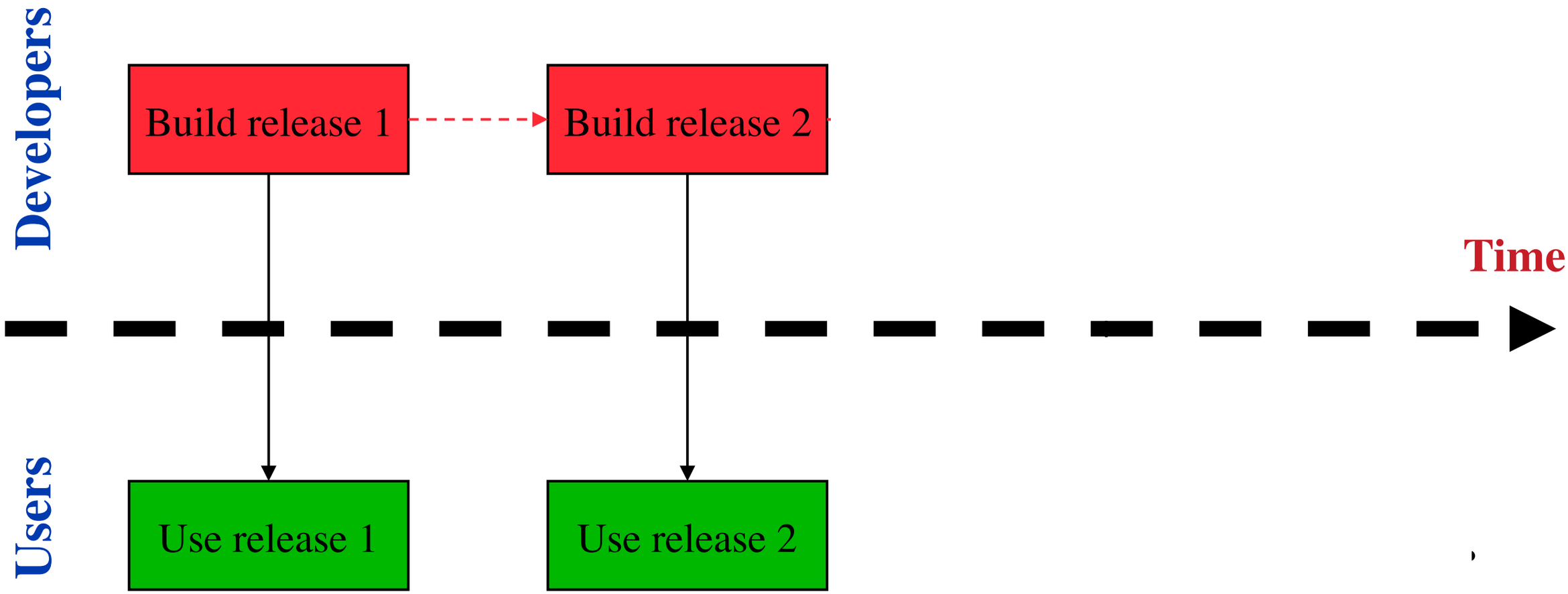
Spiral: Cons

- More appropriate for internal rather than contract development
- Relies on expertise in risk assessment
- Needs more elaboration of the different phases in a cycle

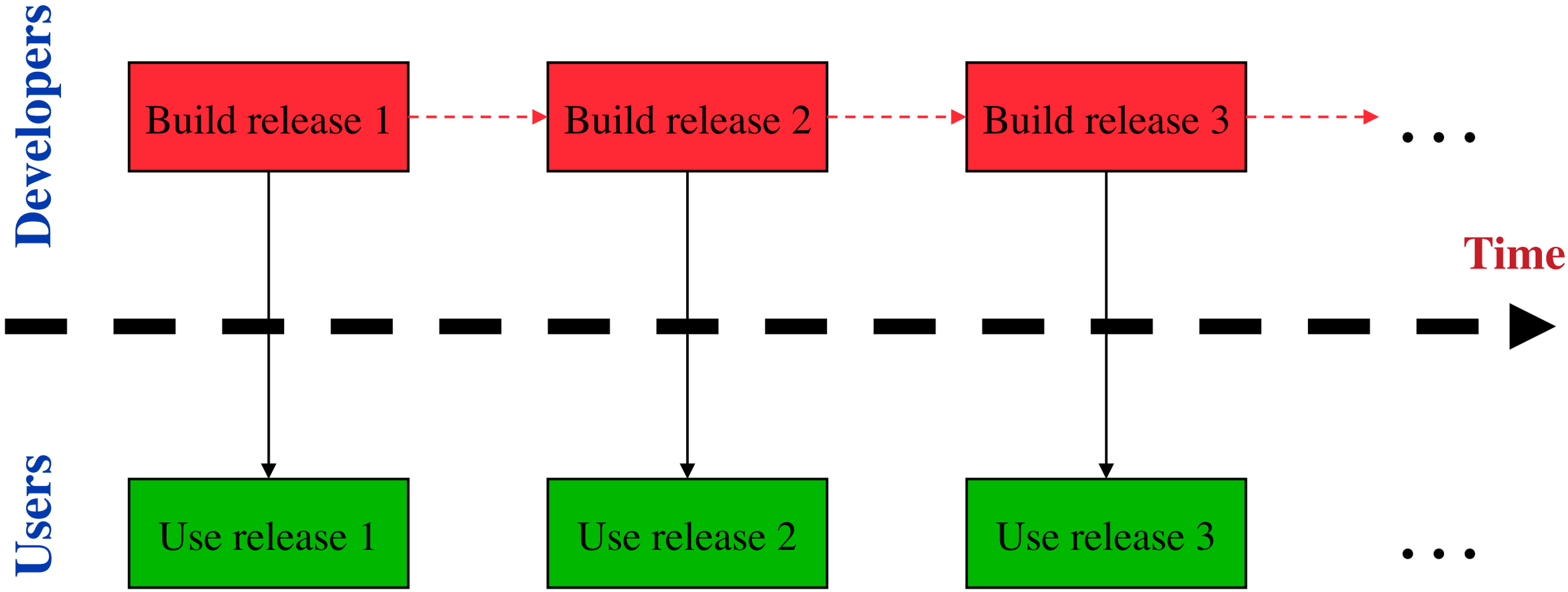
Phased Software Development Process



Phased Software Development Process

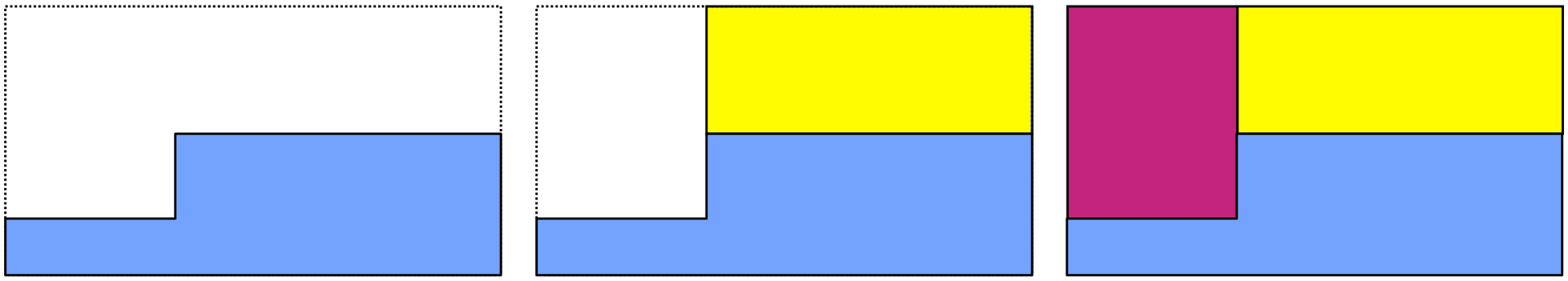


Phased Software Development Process

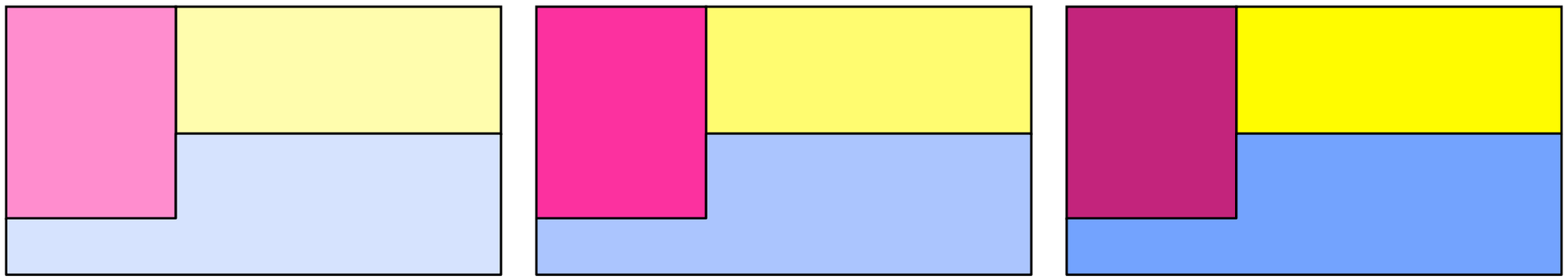


Incremental & Iterations

incremental development → partial system; full functionality



iterative development → full system; partial functionality



Phased: Pros

- Promotes system modularity
  - well-structured, robust, maintainable
- Reduces the risk of project failure
- Allows early feedback
- Can create new markets early
- Allows frequent releases
- Allows appropriate expertise to be applied

Phased: Cons

- The system pieces need to be relatively small
- It may be hard to identify common facilities needed by all pieces

Software Development Processes

- Code and Fix
- Waterfall
- Prototyping
- Spiral
- Phased
- Agile
- Open source
- Task oriented
- UP

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

http://agilemanifesto.org/

敏捷軟體開發宣言

藉著親自並協助他人進行軟體開發，
我們正致力於發掘更優良的軟體開發方法。
透過這樣的努力，我們已建立以下價值觀:

個人與互動 重於 流程與工具
可用的軟體 重於 詳盡的文件
與客戶合作 重於 合約協商
回應變化 重於 遵循計劃

也就是說，雖然右側項目有其價值，
但我們更重視左側項目。

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

http://agilemanifesto.org/principles.html

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

http://agilemanifesto.org/principles.html

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

http://agilemanifesto.org/principles.html

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

http://agilemanifesto.org/principles.html

4. Business people and developers must work together daily throughout the project.

http://agilemanifesto.org/principles.html

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

http://agilemanifesto.org/principles.html

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

http://agilemanifesto.org/principles.html

7. Working software is the primary measure of progress.

http://agilemanifesto.org/principles.html

9. Continuous attention to technical excellence and good design enhances agility.

http://agilemanifesto.org/principles.html

10. Simplicity--the art of maximizing the amount of work not done--is essential.

http://agilemanifesto.org/principles.html

11. The best architectures, requirements, and designs emerge from **self-organizing** teams.

http://agilemanifesto.org/principles.html

Agile Principles—*The Agile Manifesto*

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- over processes and tools
- over comprehensive documentation
- over contract negotiation
- over following a plan

-- http://www.agilemanifesto.org/

Agile Principles—*The Agile Manifesto*

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- over comprehensive documentation
- over contract negotiation
- over following a plan

-- http://www.agilemanifesto.org/

Agile Principles—*The Agile Manifesto*

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- over contract negotiation
- over following a plan

-- http://www.agilemanifesto.org/

Agile Principles—*The Agile Manifesto*

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- over following a plan

-- http://www.agilemanifesto.org/

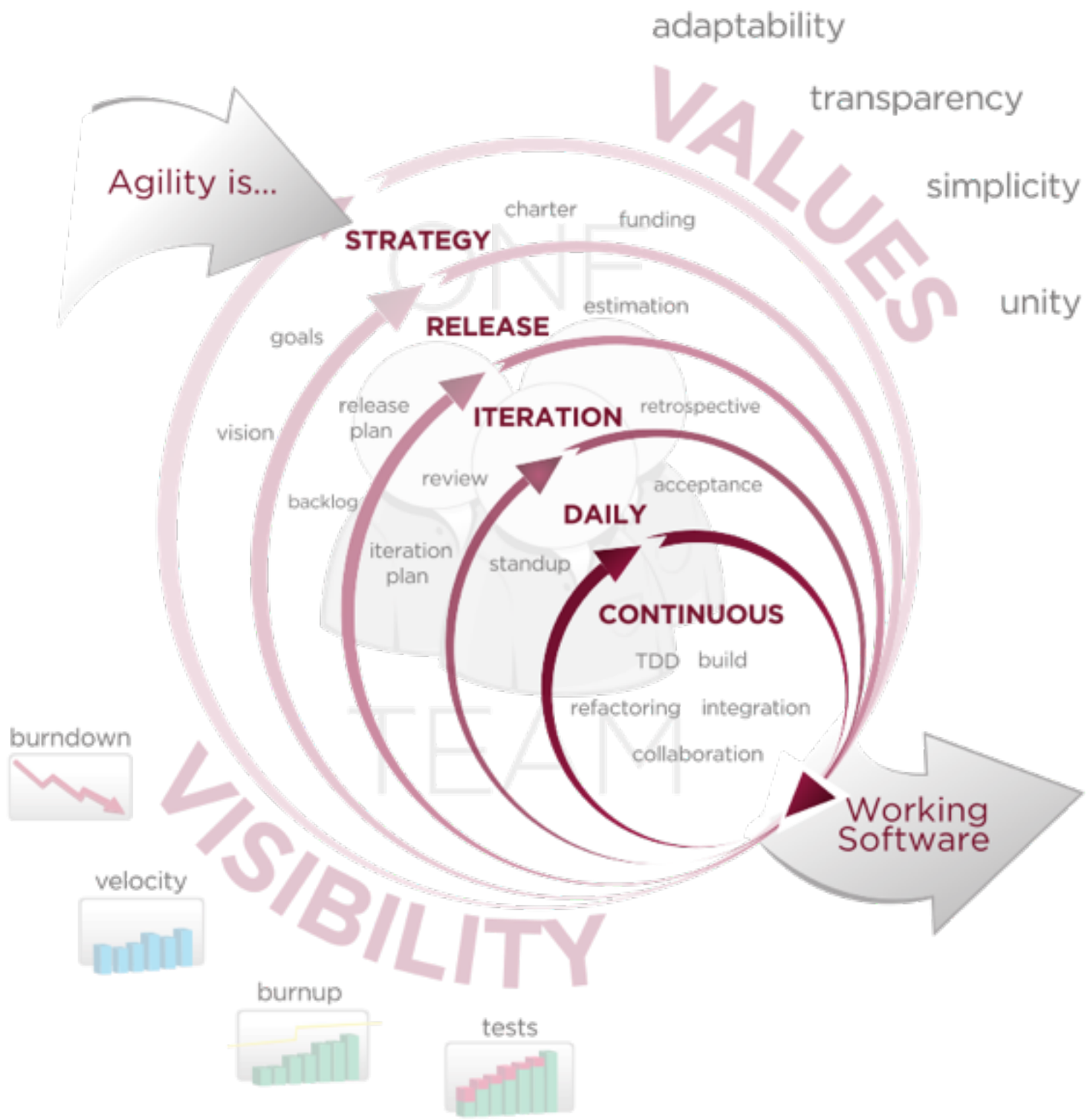
Agile Principles—*The Agile Manifesto*

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

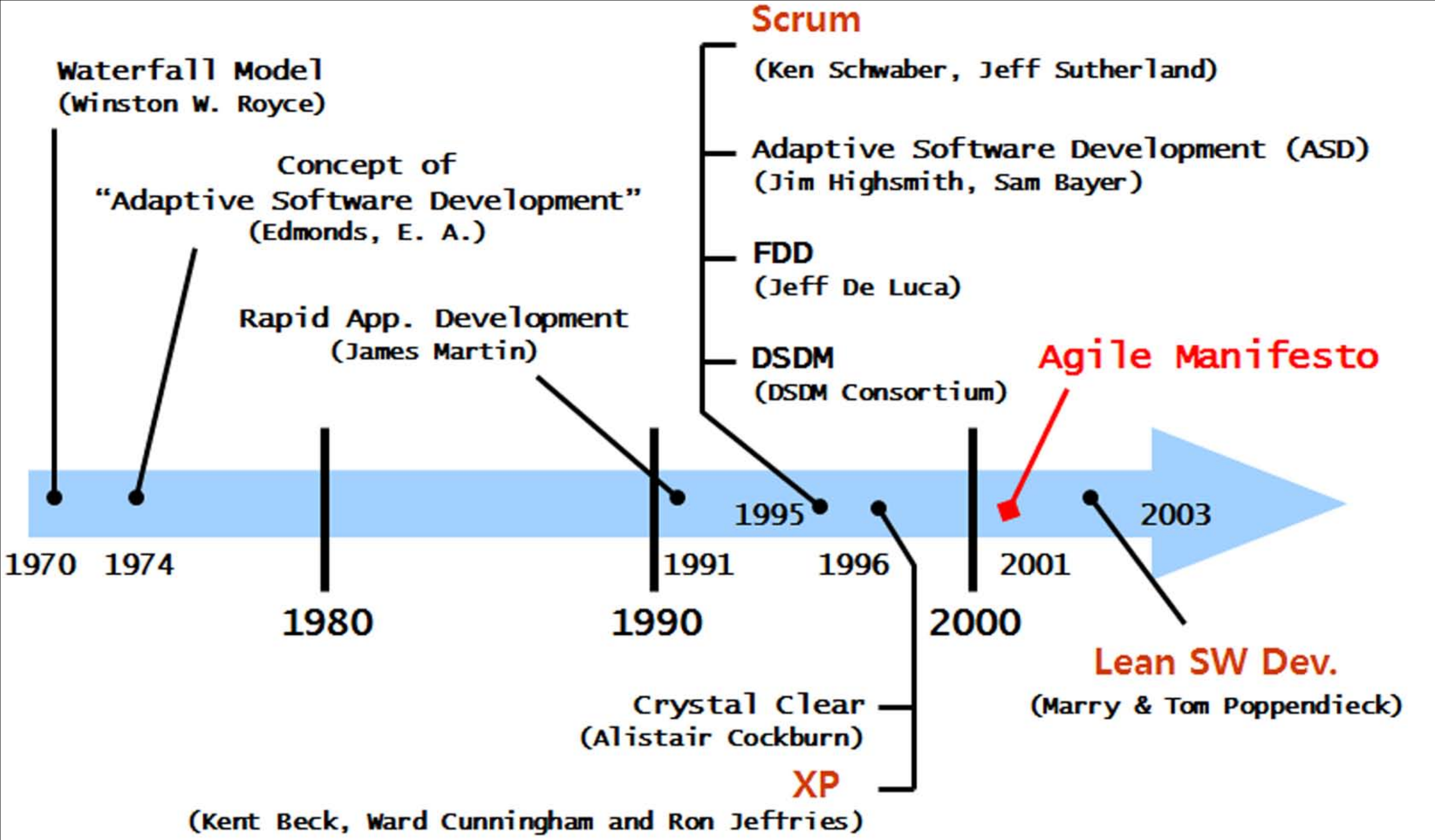
- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

-- http://www.agilemanifesto.org/

AGILE DEVELOPMENT



ACCELERATE DELIVERY

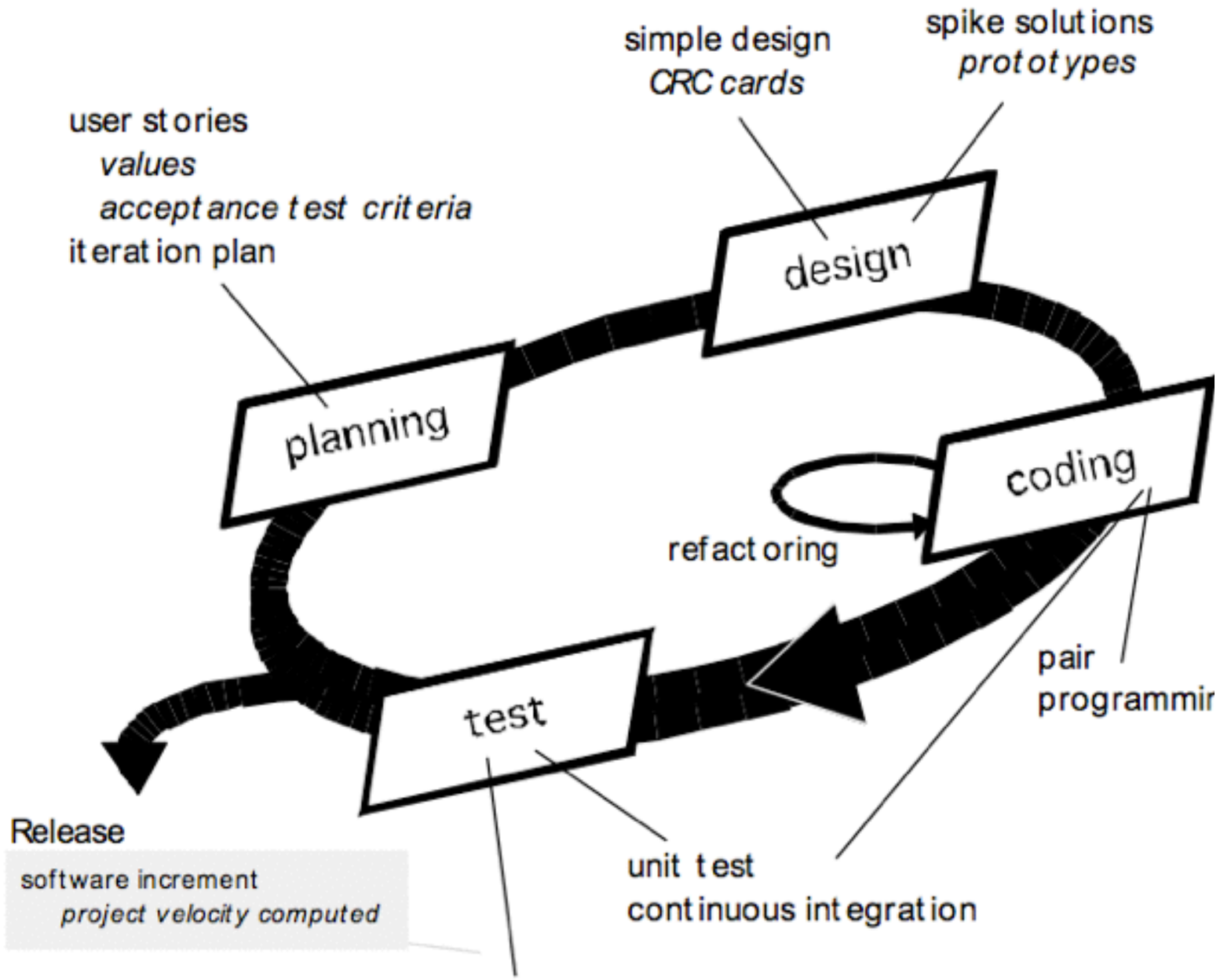


http://en.wikipedia.org/wiki/Agile_software_development

Agile Methods and Practices

- Methods
  - Extreme Programming (XP)
  - SCRUM
  - ...
- Practices
  - Pair programing
  - Test Driven Development (TDD)
  - Planning poker
  - ...

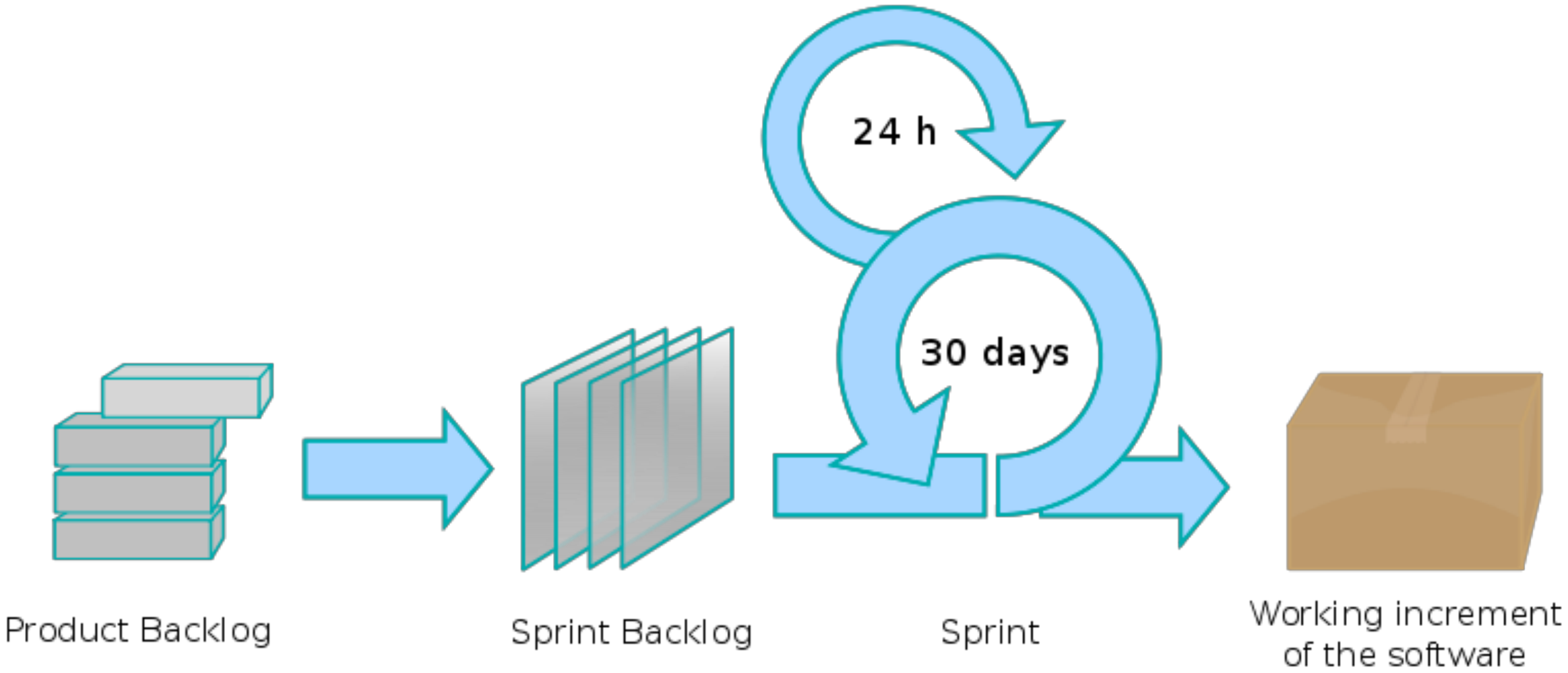
Extreme Programming (XP)



Extreme Programming (XP)

http://en.wikipedia.org/wiki/Extreme_programming

SCRUM



http://en.wikipedia.org/wiki/Scrum_%28development%29

SCRUM Tutorial

- http://www.youtube.com/watch?v=Q5k7a9YEoUI (Old)
- http://www.youtube.com/watch?v=XU0llRItyFM (New)
- http://www.youtube.com/watch?v=D8vT7G0WATM&list=PLF6BFA8BAEDF6CE70 (12 videos)

The Agile: Scrum Framework at a glance

Inputs from Executives, Team, Stakeholders, Customers, Users

Product Owner

The Team

Scrum Master

Burndown/up Charts

Daily Scrum Meeting

Every 24 Hours

1-4 Week Sprint

Sprint Review

1
2
3
4
5
6
7
8

Ranked list of what is required: features, stories, ...

Product Backlog

Team selects starting at top as much as it can commit to deliver by end of Sprint

Sprint Planning Meeting

Task Breakout

Sprint Backlog

Sprint end date and team deliverable do not change

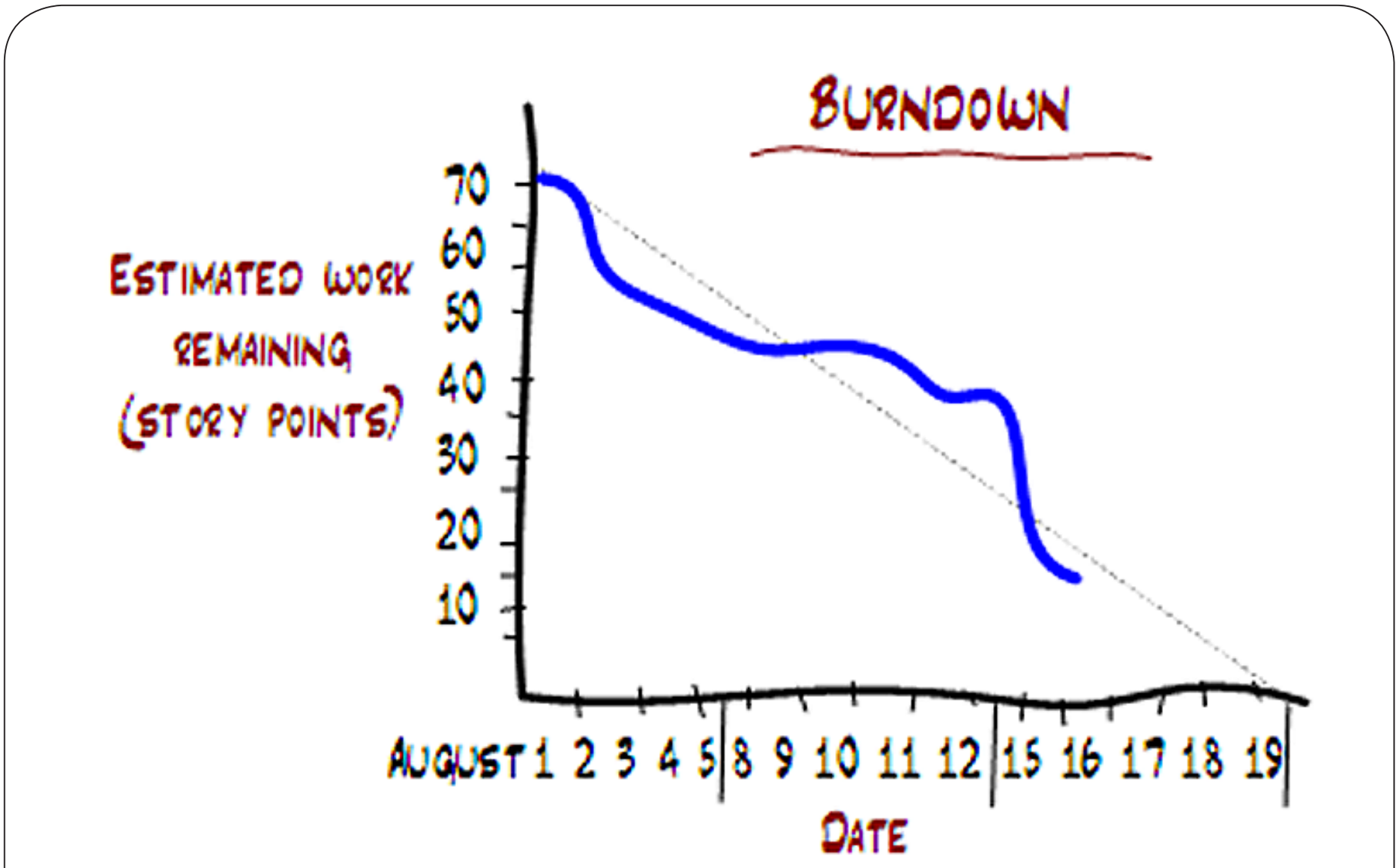
Finished Work

Sprint Retrospective

neon rain interactive

AGILE FOR ALL

Where we are?


BURNDOWN
ESTIMATED WORK REMAINING (STORY POINTS)
70
60
50
40
30
20
10
AUGUST 1 2 3 4 5 8 9 10 11 12 15 16 17 18 19
DATE

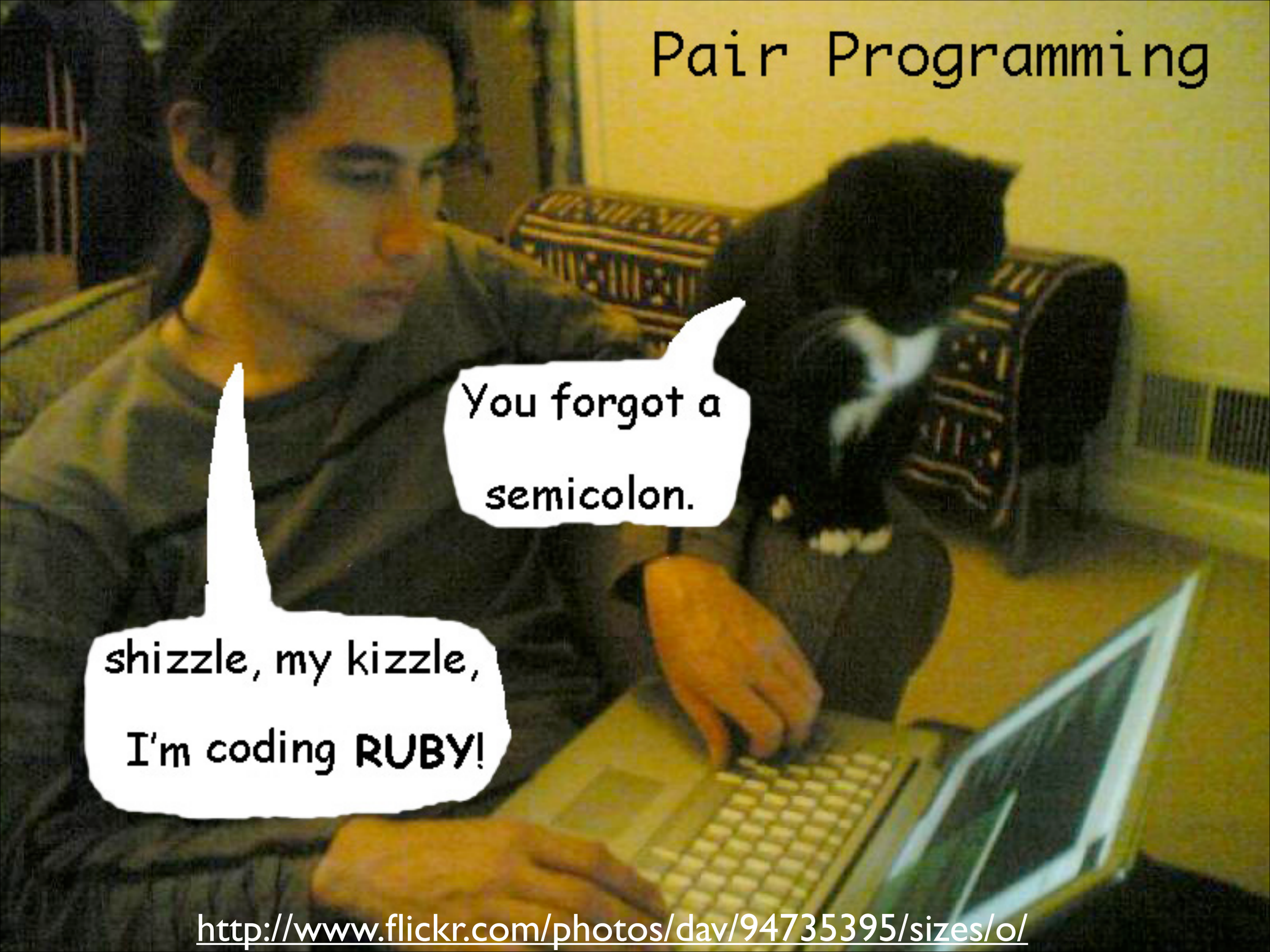
SCRUM meetings

- Sprint Planning
- Daily Scrum
- Sprint Review

Agile Methods and Practices

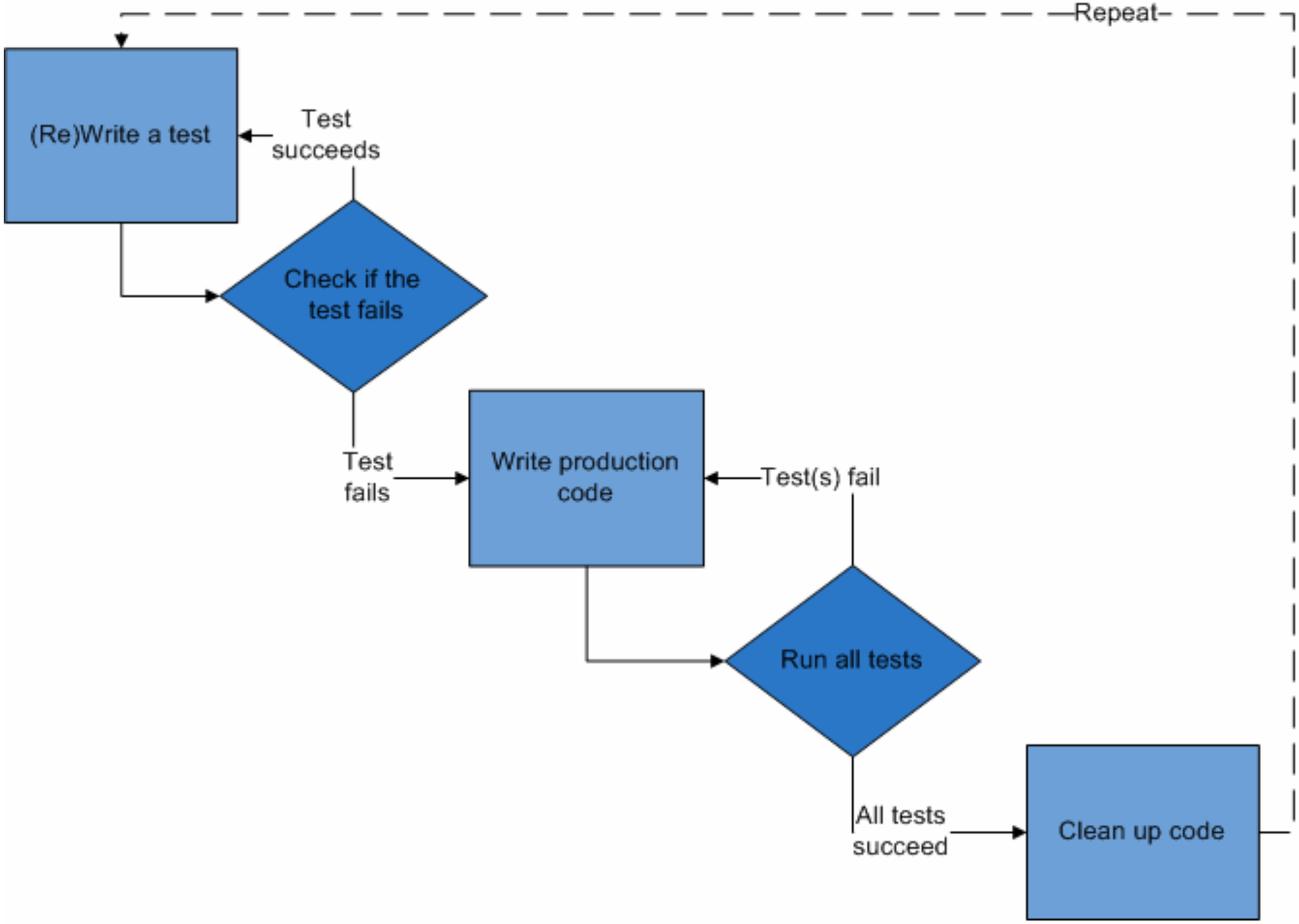
- Methods
  - Extreme Programming (XP)
  - SCRUM
  - ...
- Practices
  - Pair programing
  - Test Driven Development (TDD)
  - Planning poker
  - ...

Pair Programming



http://www.flickr.com/photos/dav/94735395/sizes/o/

TDD: Writing tests first!



My code compiles. Let's ship it!

Skeptical Developer

Write tests first before your code!

Friendly *TDD* Guru

TDD: Write tests

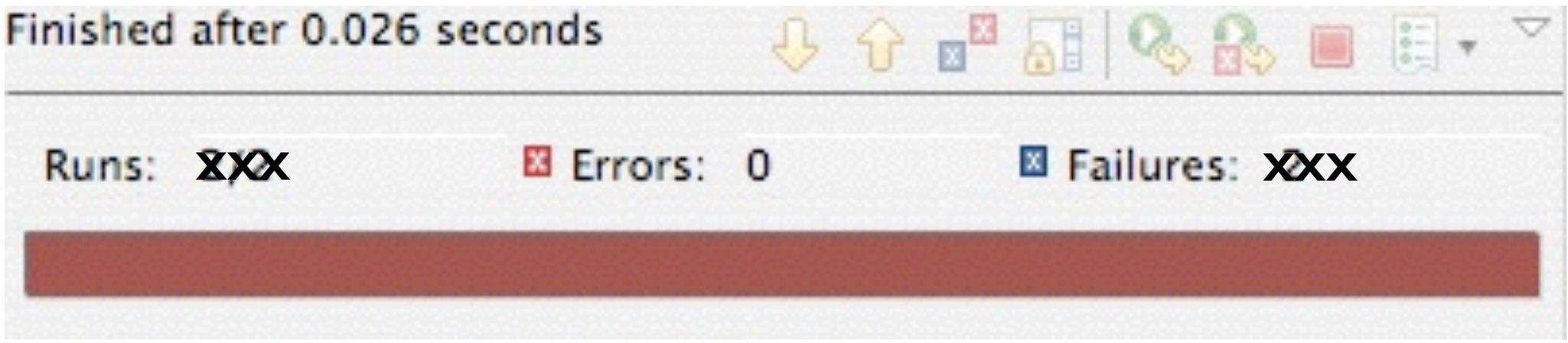
“Implement a login feature”

TDD: Write tests

```
public class LoginTestCase
{
    @Test
    public void testValidLogin()
    {
      assertTrue(Login.isValid("kim", "pass"));
    }
}
```

TDD: Run test

```
public class LoginTestCase
{
    @Test
    public void testValidLogin()
    {
      assertTrue(Login.isValid("kim", "pass"));
    }
}
```



TDD: Write code

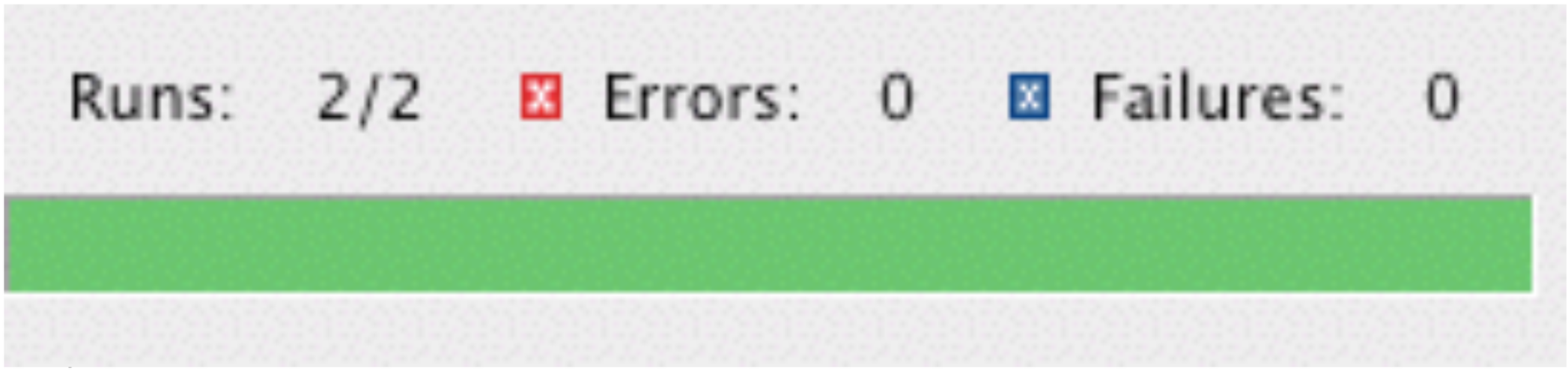
```
public class LoginTestCase
{
    @Test
    public void testValidLogin()
    {
      assertTrue(Login.isValid("kim", "pass"));
    }
}
```

```
public class Login
{
    public static boolean
            isValid(String id, String pass)
    {
      return true;
    }
}
```

TDD: Run test again

```
public class LoginTestCase
{
    @Test
    public void testValidLogin()
    {
      assertTrue(Login.isValid("kim", "pass"));
    }
}
```

```
public class Login
{
```




```
}
```

TDD: Improve test

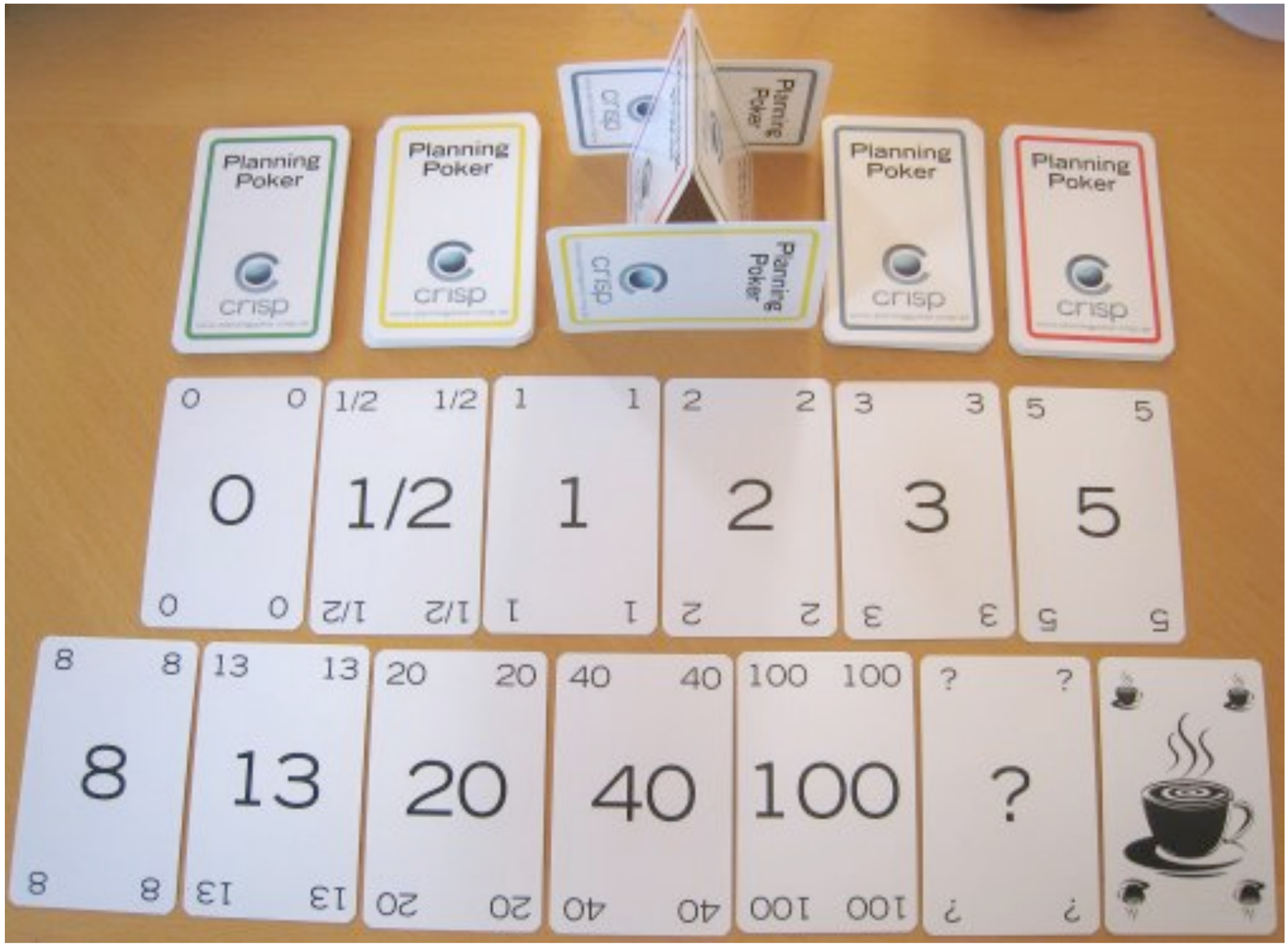
```
public class LoginTestCase
{
    @Test
    public void testValidLogin()
    {
      assertTrue(Login.isValid("kim", "pass"));
      assertFalse(Login.isValid("kim", "nopass"));
    }
}
```

```
public class Login
{
    public static boolean
            isValid(String id, String pass)
    {
      return true;
    }
}
```

Agile Methods and Practices

- Methods
  - Extreme Programming (XP)
  - SCRUM
  - ...
- Practices
  - Pair programing
  - Test Driven Development (TDD)
  - Planning poker
  - ...

Planning Poker: estimation



http://en.wikipedia.org/wiki/Planning_poker

Planning Poker

- Feature List
- A deck of cards (0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100)
- Discuss each feature and select a card
- Simultaneously show the card
- People with highest and lowest estimates justify their selection

Scrum and XP

Scrum XP

1 Onsite customer

2 User story

3 Release Planning

4 Estimation

5 Planning Poker

6 Sprint, Iteration

7 Sprint Planning

8 Daily meeting

9 Pair Programming

Task Board

10 CI

Simple Design

Refactoring

TDD

11 Velocity

Small release

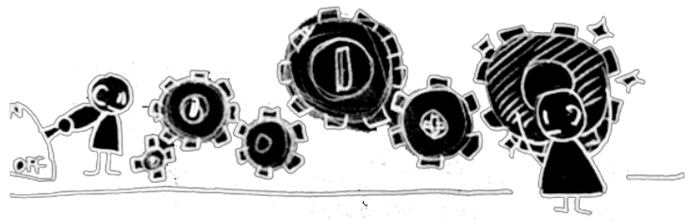
12 Burn-down chart

Sprint Review

13 Retrospective

8 Daily Meeting

10 Continuous Integration



2 User Story

3 Release Planning

12 Sprint Review

13 Retrospective

Waterfall VS Agile

Agile	Plan-driven (waterfall)
• Low criticality • Senior developers • Requirements change very often • Small number of developers • Culture that thrives on chaos	• High criticality • Junior developers • Requirements don't change too often • Large number of developers • Culture that demands order

Additional Video:

http://www.youtube.com/watch?v=gDDO3ob-4ZY

http://en.wikipedia.org/wiki/Agile_software_development

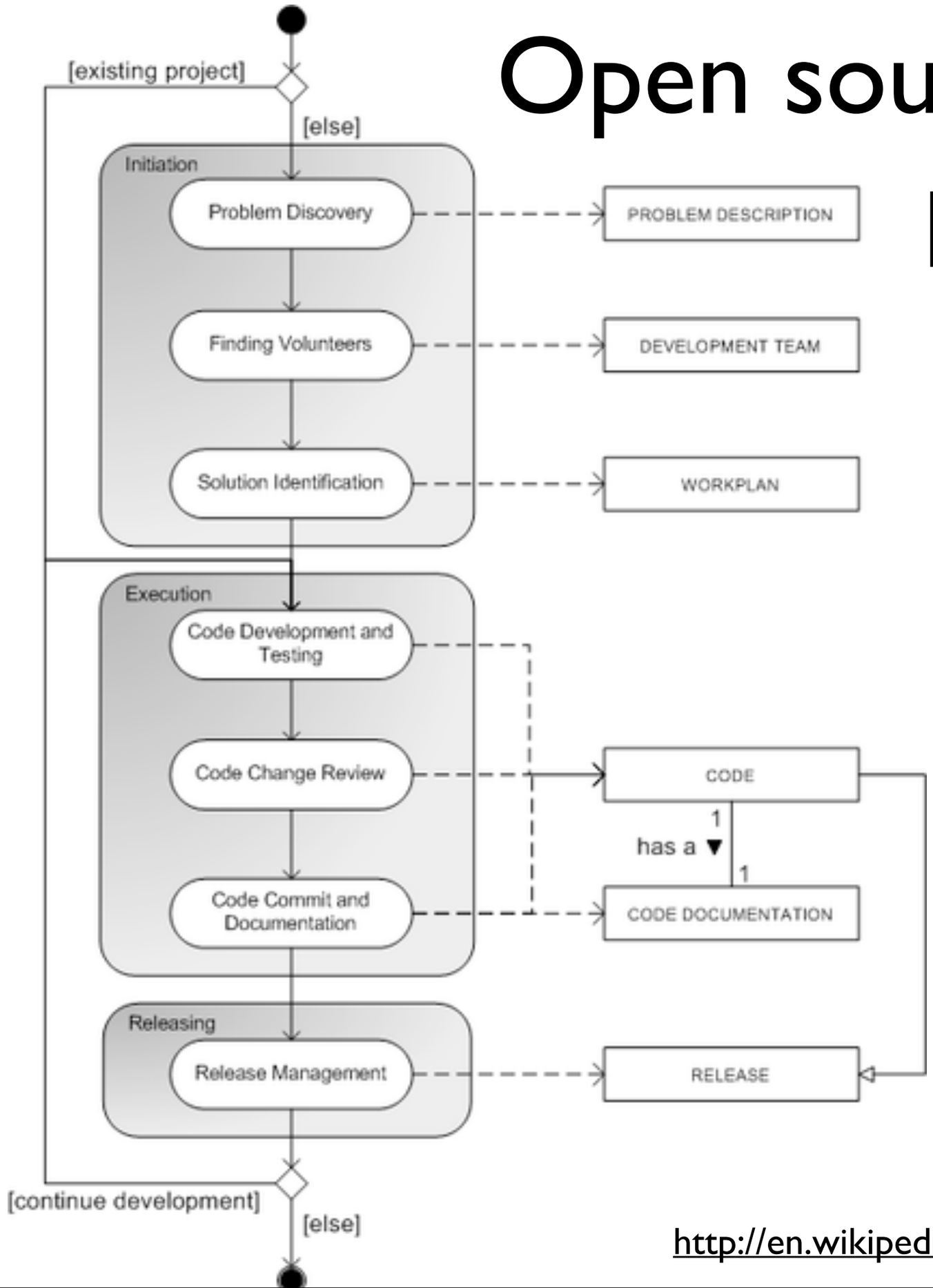
Agile? No silver bullet!

- Proven successful for small-scale and small-team projects
- Lack of structure and documents
- Require senior developers

Open source development

- No face-to-face interaction
- Developers come and leave
- Volunteer based
  - Some does not like writing documents
- Online/tool-based development

Open source development process



http://en.wikipedia.org/wiki/Open_source_software_development

Tools for open source projects

- Communication
  - IRC, emails, blogs, twitter
- Version control systems
  - CVS, SVN, Mercurial
- Issue tracker
  - Bugzilla, JIRA,
- Collaborative Development Environment
  - Wiki, Sourceforge, GoogeCode, Github

Summary

- Code and Fix
- Waterfall
- Prototyping
- Spiral
- Phased
- Agile
- Open source

Open questions

- Which SW development processes should use?
- How would evaluate the performance?