Chapter 9

Emerging Trends in software Engineering

By Yohannes S.

Contents

- Introduction
- Trends
- Managing Complexity
- Pervasive Computing
- Cloud Computing
- Emergent Requirements
- Open source
- Process Trends
- The Grand Challenge
  - Collaborative Development
  - Requirements Engineering
  - Model-Driven Development
  - Test-driven development
- Tools Trends

Introduction

- Unlike ‘Engineering industries’, software industry is about 50 years old
- **Practitioners** and **researchers** have developed an array of process models, technical methods, and automated tools in an effort to foster fundamental change in the way we build computer software
- However, past experience indicates that there is a tacit desire to find the “silver bullet”
  - the magic process or transcendent technology that will allow us to build large, complex, software based systems easily, without confusion, without mistakes, without delay, without the many problems

Trends

- No one can predict the future with absolute certainty
- But it is possible to assess trends in the software engineering area and from those trends to suggest possible directions for the technology
- Software intensive systems (SIS) have become the foundation of virtually every modern technology
  - Software content in virtually every product and service will **continue to grow** —in some cases dramatically
  - Software must be demonstrably **safe**, **secure**, and **reliable**
  - **Requirements** will **emerge** as systems **evolve**

Trends…

- The trends that have an effect on software engineering technology often come from the **business**, **organizational**, **market**, and **cultural** arenas.
- These "**soft trends**" can guide the direction of research and the technology that is derived as a consequence of research
- **Soft Trends**
  - The broad characteristics of the new systems we build
  - The anthropological and sociological characteristics of the new generation of people who do software engineering work
- **Hard Trends**
  - The technical aspects of next generation

Soft Trends…

- *Connectivity and collaboration* (enabled by high bandwidth communication)
  - has already led to a software teams that do not occupy the same physical space (telecommuting and part-time employment in a local context)
- *Globalization*
  - leads to a diverse workforce (in terms of language, culture, problem resolution, management philosophy, communication priorities, and person-to-person interaction).
- *An aging population*
  - implies that many experienced software engineers and managers will be leaving the field over the coming decade.
  - The software engineering community must respond

Soft Trends…

- *Consumer spending in emerging economies* will double to well over $9 trillion.
  - a non-trivial percentage of this spending will be applied to products and services that have a **digital component**—that are software-based or software-driven

- People and Teams
  - As **systems grow in size**, **teams grow** in number, geographical distribution, and culture
  - As systems **grow in complexity**, **team interfaces** become pivotal to success
  - As systems become **pervasive**, teams must manage **emergent requirements**
  - As systems become more open, *what is a team?*

Managing Complexity

- In the relatively near future, systems requiring over 1 billion LOC will begin to emerge
- Consider the interfaces for a billion LOC system
  - to the outside world
  - to other interoperable systems
  - to the Internet (or its successor), and
  - to the millions of internal components that must all work together to make this computing monster operate successfully.
- Is there a reliable way to ensure that all of these connections will allow information to flow properly?
- Consider the project itself
  - Consider the number of people (and their locations) who will be doing the work

Pervasive Computing (PvC)

- Concepts such as
  - **ambient intelligence,**
  - **context-aware applications,** and
  - **pervasive/ubiquitous computing**

  all focus on integrating software-based systems into an environment far broader than anything to date

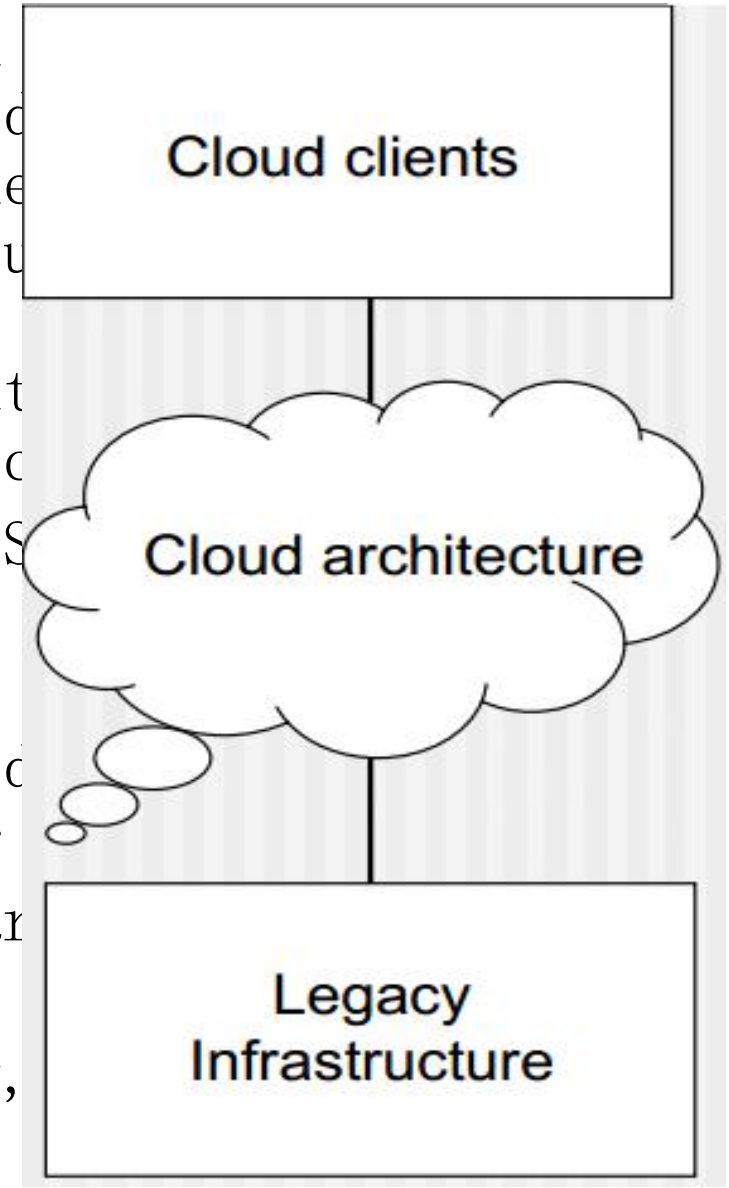
- **open-world software**
  - software that is designed to adapt to a continually changing environment ‘by self-organizing its structure and self-adapting its behavior

Pervasive Computing (PvC)…

- First stage (PvC-1) [*today*]
  - Device mobility and ad hoc networking
  - Simple context awareness
  - *Soon:* smart objects implemented in devices that have the potential to communicate with one another
- Second stage (PvC-2) *[over the next decade]*
  - Mobile user profiles that can be recognized by other objects
  - Smart objects will respond to other objects based on situational characteristics
- Testing issues
  - Considerable environmental variation
  - Complex communication issues
  - Adaptive processing requirements

Cloud Computing

- Cloud Computing is a paradigm in
  information is permanently stored
  servers on the Internet and cache
  temporarily on clients that inclu
  desktops, entertainment centers,
  computers, notebooks, wall comput
  handhelds, sensors, monitors, etc
- Provides software as a service (S
- Device and location independence
  enables users to access systems
  regardless of their location or c
- Multi-tenancy enables sharing of
  resources (and costs) among a lar
  of users
- Demands reliability, scalability,
  sustainability (Green IT)



Emergent Requirements

- As systems become more complex, requirements will emerge as everyone learns more about it,
  - The system's interoperable elements
  - The environment in which it is to reside, and
  - The objects that interact with it
- This reality implies a number of software engineering trends.
  - **Process models** must be designed to embrace change and adopt the basic tenets of the agile philosophy
  - **Methods** that yield engineering models (e.g., requirements and design models) must be used judiciously because those models will change repeatedly as more knowledge about the system is acquired

Open source

- *Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.*

- The term ***open source*** when applied to computer software, implies that software engineering work products (models, source code, test suites) are open to the public and can be reviewed and extended (with controls) by

Process Trends

- **SPI frameworks** – will emphasize "strategies that focus on goal orientation and product innovation."
- **Process changes** will be driven by the needs of practitioners and should start from the bottom up
- Greater emphasis will be placed on the **return-on-investment of SPI activities**
- **Expertise in sociology and anthropology** may have as much or more to do with successful SPI as other, more technical disciplines.
- **New modes of learning** may facilitate the transition to a more effective software process.
- **Automated software process technology (SPT)** will move away from global process management (broad-based support of the entire software process) to

The Grand Challenge

- There is one trend that is undeniable—software-based systems will undoubtedly become **bigger and more complex** as time passes.
- It is the engineering of these large, complex systems, regardless of delivery platform or application domain, the poses the “grand challenge” for software engineers.
- Key approaches:
  - more effective distributed and collaborative software engineering philosophy
  - better requirements engineering approaches
  - a more robust approach to model-driven development, and

The Grand Challenge…

- Collaborative Development
  - Today, software engineers collaborate across time zones and international boundaries, and every one of them must share information.
  - The challenge over the next decade is to develop methods and tools that facilitate that collaboration.
  - Critical success factors:
    - Shared goals
    - Shared culture
    - Shared process
    - Shared responsibility

The Grand Challenge…

- **Requirements Engineering**
  - To improve the manner in which requirements are defined, the software engineering community will likely implement three distinct sub-processes as RE is conducted
    - **improved knowledge acquisition and knowledge sharing** that allows more complete understanding of application domain constraints and stakeholder needs
    - greater emphasis on **iteration** as requirements are defined
    - more **effective communication and coordination tools** that enable all stakeholders to collaborate effectively.

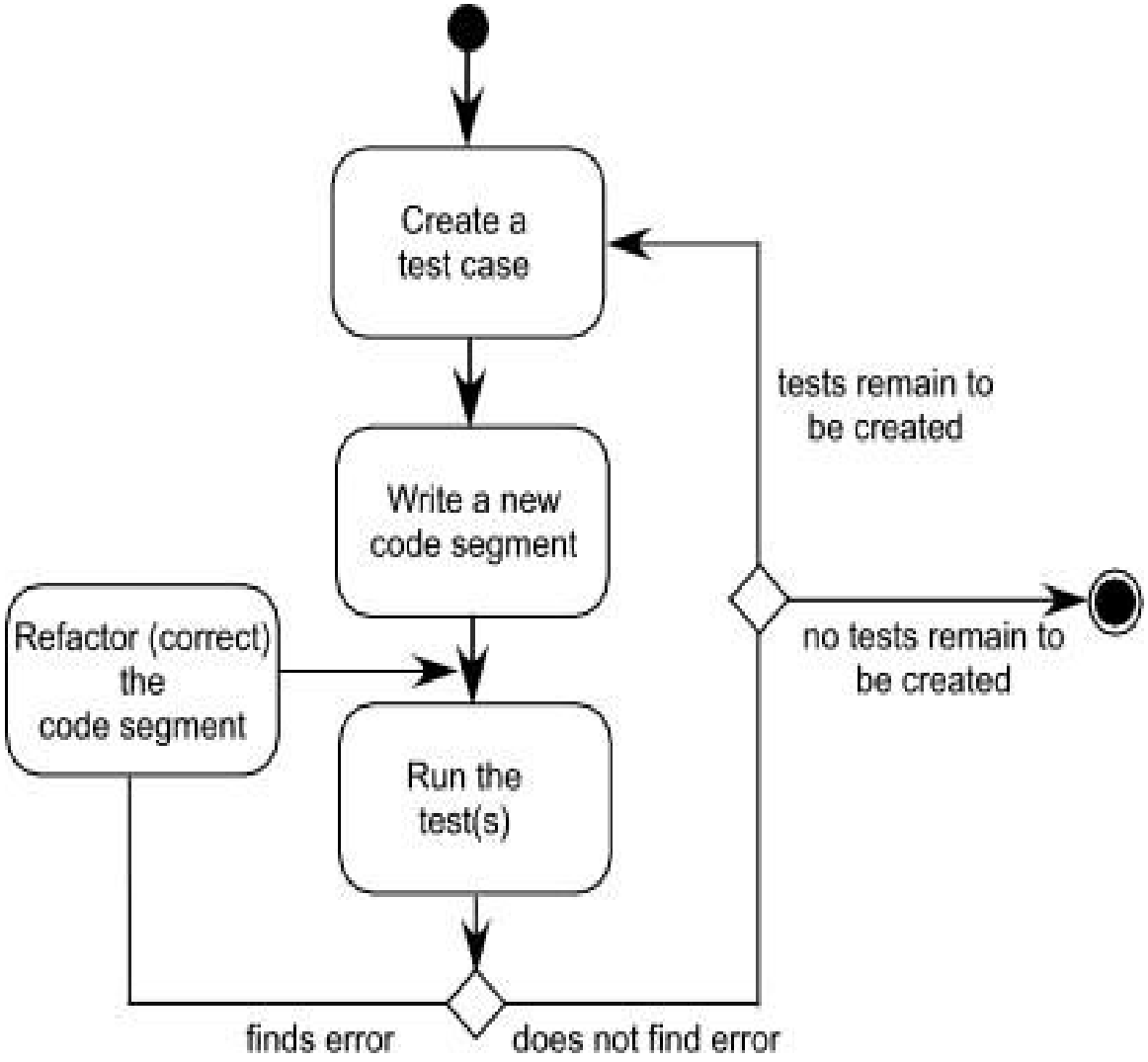
The Grand Challenge…

- Model-Driven Development
  - Couples domain-specific modeling languages with transformation engines and generators in a way that facilitates the representation of abstraction at high levels and then transforms it into lower levels

  - *Domain-specific modeling languages* (DSMLs)
    - represent "application structure, behavior and requirements within particular application domains"
    - described with **meta-models** that "define the relationships among concepts in the domain and precisely specify the key semantics and constraints associated with these domain concepts."

The Grand Challenge…

- Test-driven development (TDD)
  - requirements for a software component serve as the basis for the creation of a series of test cases that exercise the interface and attempt to find errors in the data structures and functionality delivered by the component.
  - TDD is not really a new technology but rather a trend



Tools Trends

- **Requirements engineering tools** will combine voice recognition input with "text mining" to extract requirements from informal information sources

- As pervasive computing becomes commonplace, **design modeling tools** must allow the designer to consider the architecture and behavior of the software and the physical properties of the devices on which the software resides.

- As test-driven development approaches gain momentum, **tools for selecting test cases**