# Large Scale Software Development

Oleh

Bayu Hendradjaya

# What are The Problems in Large Size Application?

- Memory Constraint
  - Memory has to be managed carefully
    - System hardware memory constraint
- Time constraint
  - Waiting for rebuild (compile time)
- Performance
  - Java programs are an interpreted language, performance will never catch up with compiled language
  - Garbage collection handling
- Threads
  - Problem caused by threading is much more complex than handling a sequential programming
- Safety
  - The bigger the problem is, the riskier it becomes
- Portability
  - Thread scheduling can vary from platform to platform
  - Platform specific notation (c:\meta\test.java to \meta\test.java )
  - Different machine may introduce different bugs in their java compilers/VM
  - Non standard API

## Development in a large scale software is not just the issue of the large size software application

## Typical Large Scale Software

- Large quantities of source code
  - typically millions of lines
- Large numbers of developers
  - potentially hundreds, often geographically distributed
- High complexity of interaction between components
- Extensive use of off-the-shelf components
- Multiple programming languages
- Multiple persistence mechanisms
  - files, relational databases, object databases
- Multiple hardware platforms
- Distribution of components over several hardware platforms
- High concurrency

## Large Scale (LS) Software Development

- Complexity in “Software”
  - High LOC, High amount of Data Acces, High computation elements, etc
- Complexity in Requirements
  - High Requirements Size, Requirements Changes, conflicting requirements, etc
- Complexity in location differences
- Complexity in platform differences
- Complexity in “politics”
  - Different rules, policies
- Complexity in Resource Allocations

## Principles of Large-Scale Software Development

- Reproducibility
- Policy Enforcement versus Policy Auditing
- Process, Process, Process

  A process must be defined to be improved
- Automation

  A process must be automated to be repeatable
- Continuous Functional Integration and Test

## Coordination in LS SW Development

- The root cause is work coordination between team developers
  - intra- and inter- team coordination
- Major factors are
  - Communication
  - Capacity
  - Cooperation

## An example of Large Scale Software Development

## Example of Large Systems

- At Syracuse at least two very large software systems have been successfully implemented:
  - Over The Horizon Radar (OTHR) contained about 3,000,000 lines of code, written mostly in FORTRAN with some C and some assembly language as well.
  - BSY-2 submarine battle management system software is several times larger than OTHR, written mostly in Ada.
- A 5 million line system would require something like:

$$\frac{5{,}000{,}000 lines}{22 lines / day} x \frac{1}{240 days / year} = 947\, person\, years of\, effort$$

## Large Systems (2)

- To complete the project in 2.5 years would require the services of at least 380 well trained software developers.
- Two points are almost self evident:
  - a system this large must be partitioned into many relatively small, nearly independent components in order to get it to work
  - it would be much better not to create such a large system at all, but rather, to build most of it from reused software components, reserving new code for new requirements.

## Another example of Large Scale Software Development

## HP-UX Development

- 220,000+ Source Files
- ~2.5 Million Versions
- 52,000+ Derived Files (Single Nightly Build)
- 900+ User Accounts
- More Than 1 Terabyte of Disk Space

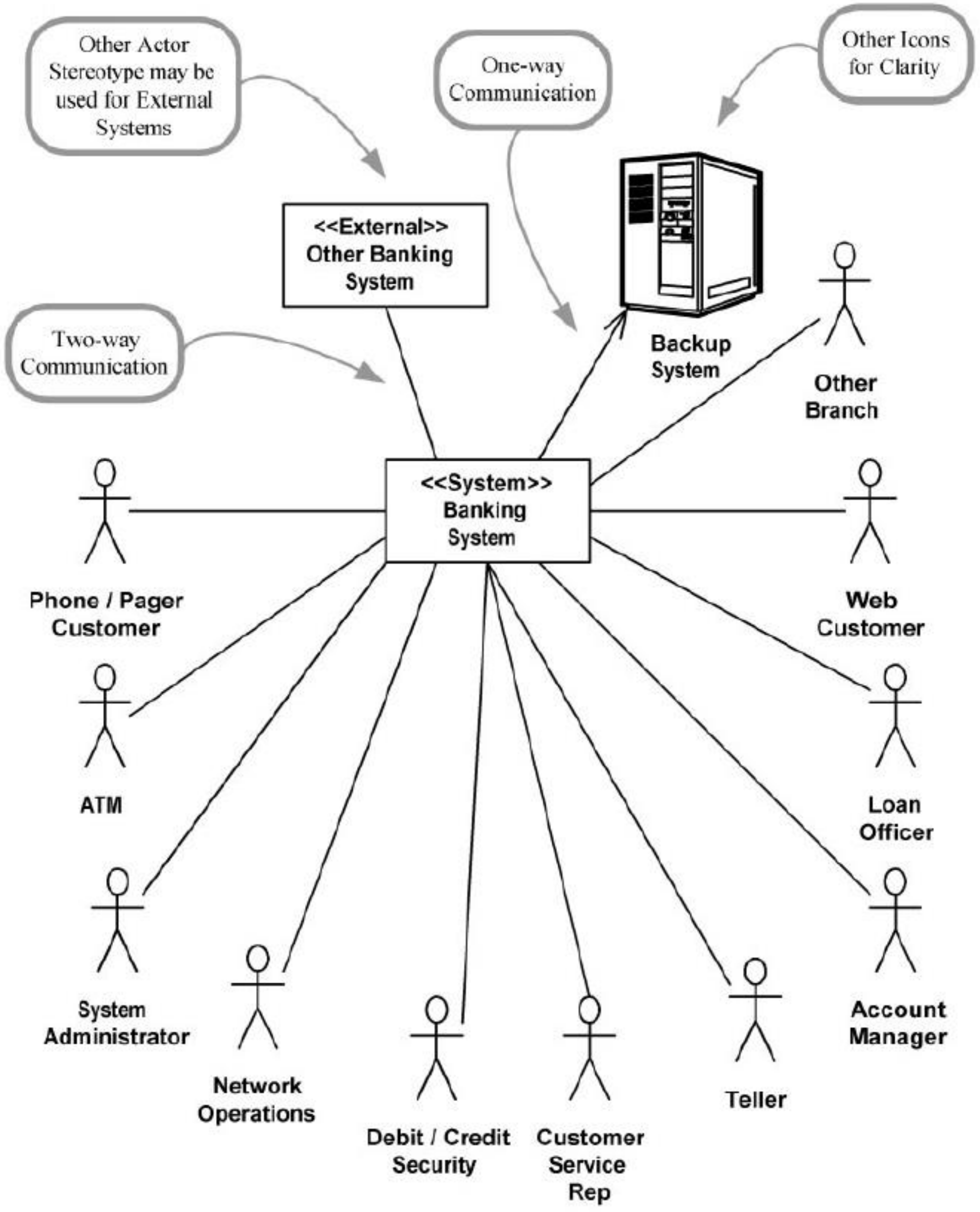
## HP-UX Concurrent Development

- Normally 2-3 Releases Under Development
- Several Releases Under Maintenance
- Heavy Use of Branching and Merging

## HP-UX Multisite Development

- Four Primary Development Sites
  - Three in USA (Three Time Zones)
  - One in India
- One USA Site Hosts Three Logical Sites
  - Partner Companies
- Several Secondary Sites

# Not so large but large enough



# Very Large-Scale Software Engineering (VLSSE)

- Architectural specification (system, services (data, protocols, processes), team/organization.
- Configuration management and version control
- Process automation—builds, regression tests, etc.
- Reliance on *informalisms* for development

(Open Source Software Development and Very Large-Scale Software Engineering (Walt Scacchi *Institute for Software Research University of California, Irvine Irvine, CA, 92697-3425 USA 19 July 2005*))

## VLSSE - The Process

- Configuration management and version control
- Architectural specification and maintenance
- Collaboration, leadership, control and conflict management
- Community development and support
- Software source code and artifact data mining

## Ultra Large Scale Software Development

# Ultra Large Scale (ULS) System

- **Ultra-large-scale system (ULSS)** is a term used in fields including Computer Science, Software Engineering and Systems Engineering to refer to software intensive systems with unprecedented amounts of hardware, lines of source code, numbers of users, and volumes of data.
- The scale of these systems gives rise to many problems:
  - they will be developed and used by many stakeholders across multiple organizations, often with conflicting purposes and needs;
  - they will be constructed from heterogeneous parts with complex dependencies and emergent properties;
  - they will be continuously evolving; and software, hardware and human failures will be the norm, not the exception.

(http://en.wikipedia.org/wiki/Ultra-large-scale_systems)

# ULS System – Characteristics

## Ultra-Large-Scale Systems; The Software Challenge of the Future (Software Engineering Institute – Carnegie Mellon)

- **Decentralization**
  - The scale of ULS systems means that they will necessarily be decentralized in a variety of ways—decentralized data, development, evolution, and operational control.
- **Inherently conflicting, unknowable, and diverse requirements**
  - ULS systems will be developed and used by a wide variety of stakeholders with unavoidably different, conflicting, complex, and changing needs.
- **Continuous evolution and deployment:**
  - There will be an increasing need to integrate new capabilities into a ULS system while it is operating. New and different capabilities will be deployed, and unused capabilities will be dropped; the system will be evolving not in phases, but continuously.

## ULS System – Characteristics (2)

- **Heterogeneous, inconsistent, and changing elements:**
  - A ULS system will not be constructed from uniform parts: there will be some misfits, especially as the system is extended and repaired.
- **Erosion of the people/system boundary**
  - People will not just be users of a ULS system; they will be elements of the system, affecting its overall emergent behavior.
- **Normal failures**
  - Software and hardware failures will be the norm rather than the exception.
- **New paradigms for acquisition and policy**
  - The acquisition of a ULS system will be simultaneous with the operation of the system and require new methods for control.

# ULS – Challenges

**Ultra-Large-Scale Systems; The Software Challenge of the Future (Software Engineering Institute – Carnegie Mellon)**

- Design and Evolution
- Orchestration and Control
- Monitoring and Assessment

# ULS – Challenges (2)

- Design and Evolution

  The challenge will be to find new ways to harness and coordinate the design capabilities and motivations not just of individual companies, prime contractors, and supply chains, but of whole industries, within which competition for value will drive much richer and more economical exploration of complex ***design spaces.***

# ULS – Challenges (3)

- Orchestration and Control
  - By ***orchestration we mean the set of activities needed to make the elements*** of a ULS system work in reasonable harmony to ensure continuous satisfaction of the mission objectives. Orchestration involves management and administration but at a scale well beyond that of traditional, centralized, relatively fine-grained controls. Orchestration requires a combination of up-front design, overall policy promulgation and enforcement, and real-time adjustment of operating parameters.
  - Orchestrating a ULS system requires supporting interdependencies and controlling the consequences of local actions with respect to their effect on the emergent whole, even though each part of a system might be acting to maximize its local utility.

## ULS – Challenges (4)

– To succeed in developing and operating ULS systems, new knowledge, technologies, and methods in the following areas are needed:
  - online modification
  - maintenance of quality of service
  - creation and execution of policies and rules
  - adaptation to users and contexts
  - enabling of user-controlled orchestration

## ULS – Challenges (5)

- Monitoring and assessment
  - The effectiveness of ULS system design, evolution, and orchestration has to be evaluated. There must be an ability to monitor and assess ULS system state, behavior, and overall health and well being.
  - The criteria for success or overall health are different for ULS systems than for smaller systems designed to accomplish a task that does not change as the system is used.

## Domains in which ULS Systems are emerging http://www.sei.cmu.edu/uls/

- **Defense**

  The Northrop report argued that "the U.S. Department of Defense (DoD) has a goal of information dominance … this goal depends on increasingly complex systems characterized by thousands of platforms, sensors, decision nodes, weapons, and warfighters connected through heterogeneous wired and wireless networks. … These systems will push far beyond the size of today's systems by every measure … They will be ultra-large-scale systems."[1]

## Domains … (2)

- **Financial trading**

  Following the flash crash, Cliff and Northrop[2] have argued "The very high degree of interconnectedness in the global markets means that entire trading systems, implemented and managed separately by independent organizations, can rightfully be considered as significant constituent entities in the larger global super-system. … The sheer number of human agents and computer systems connected within the global financial-markets system-of-systems is so large that it is an instance of an ultra-large-scale system, and that largeness-of-scale has significant effects on the nature of the system".[2]

## Domains.... (3)

- **Healthcare**

  Kevin Sullivan has stated that the US healthcare system is "clearly an ultra-large-scale system"[7] and that building national scale cyber-infrastructure for healthcare "demands not just a rigorous, modern software and systems engineering effort, but an approach at the cutting edge of our understanding of information processing systems and their development and deployment in complex socio-technical environments".[7]
- **Others**

  Other domains said to be seeing the rise of ultra-large-scale systems include government, transport systems (for example air traffic control systems), energy distribution systems (for example smart grids) and large enterprises.

## Further Reading

- SEI CMU ULS - http://www.sei.cmu.edu/uls/
- SEI CMU, Does Scale Really Matter? Ultra-Large-Scale Systems Seven Years After the Study, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=68596
- Ultra-Large-Scale Systems: The Software Challenge of the Future, https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30519

## Issues in Scrum for a Large Team

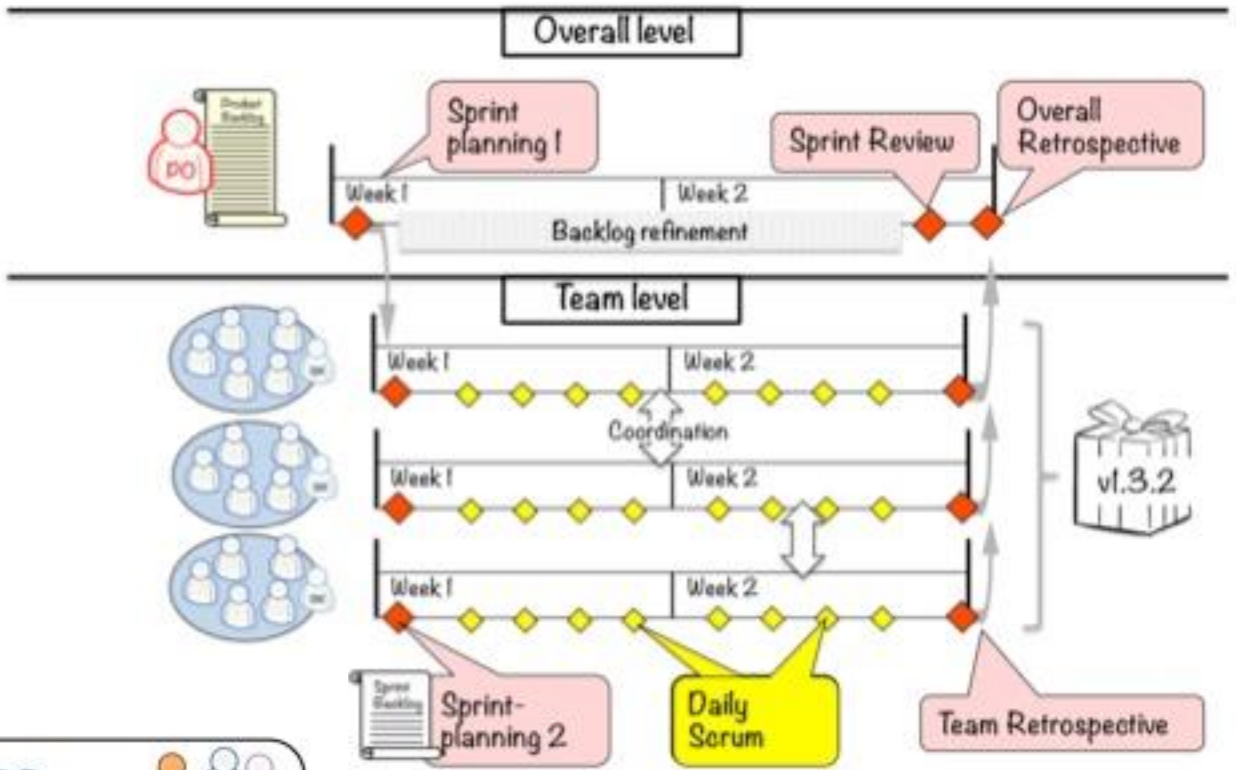
- Cross team dependencies
- Risks that affect several teams
- Scheduling of (coordinated) deliveries

# Scrum at a Large Team

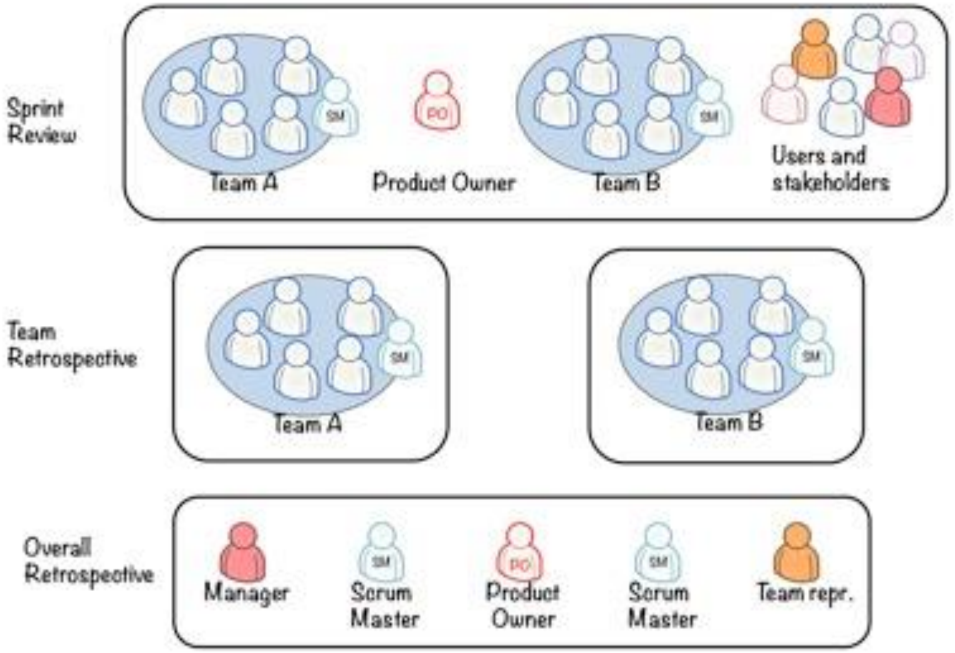
- Scaled Agile works, and using a scaling framework help you get a quick start.
- All scaling frameworks share some common patterns: Scrum at team level, many teams sharing a backlog, planning is done collaboratively across teams, and the general principles of pull and self-organization.

# LeSS (Large Scale Scrum)

- LeSS is a scaling framework that comes from Craig Larman and Bas Vodde, and is based on their work in the financial and telecommunication industries.
- LeSS is defined by minimalism and minimal process, i.e. use as little process as possible to get multiple Scrum teams to work well.
- At its most basic level, LeSS is a clarification of how Scrum is intended to work.
- Scrum teams should be long-lived cross-functional feature teams that are maintained across multiple projects
- Manage the complexity of large-scale development by simplifying it as much as possible
- LeSS recommends that multiple teams has the same Product Owner and a shared Product Backlog. Their sprints are synchronized to the Product-level Sprint leading to one integrated Potentially Shippable Product Increment. Sprint Planning, Sprint Review and Sprint Retrospective runs at the same time for all teams.

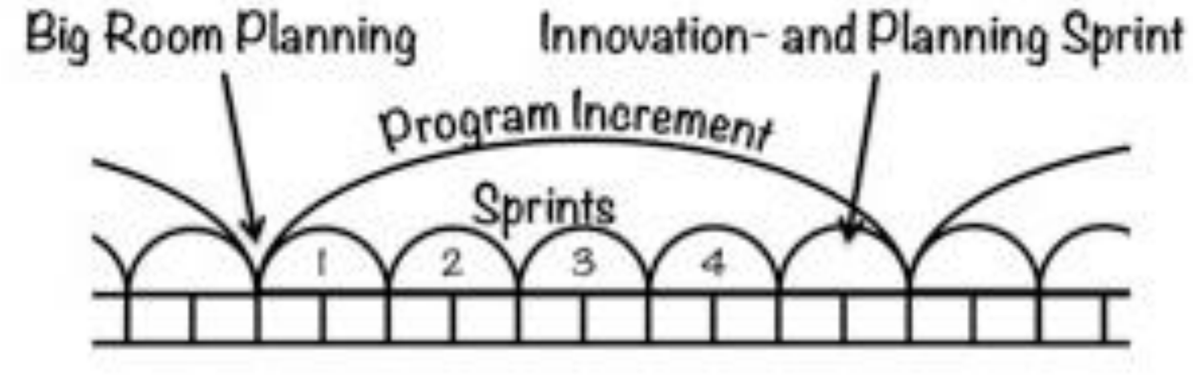


Overall level
Product Backlog
PO
Sprint planning 1
Sprint Review
Overall Retrospective
Week 1
Week 2
Backlog refinement
Team level
Week 1
Week 2
Coordination
v1.3.2
Sprint Backlog
Sprint-planning 2
Daily Scrum
Team Retrospective

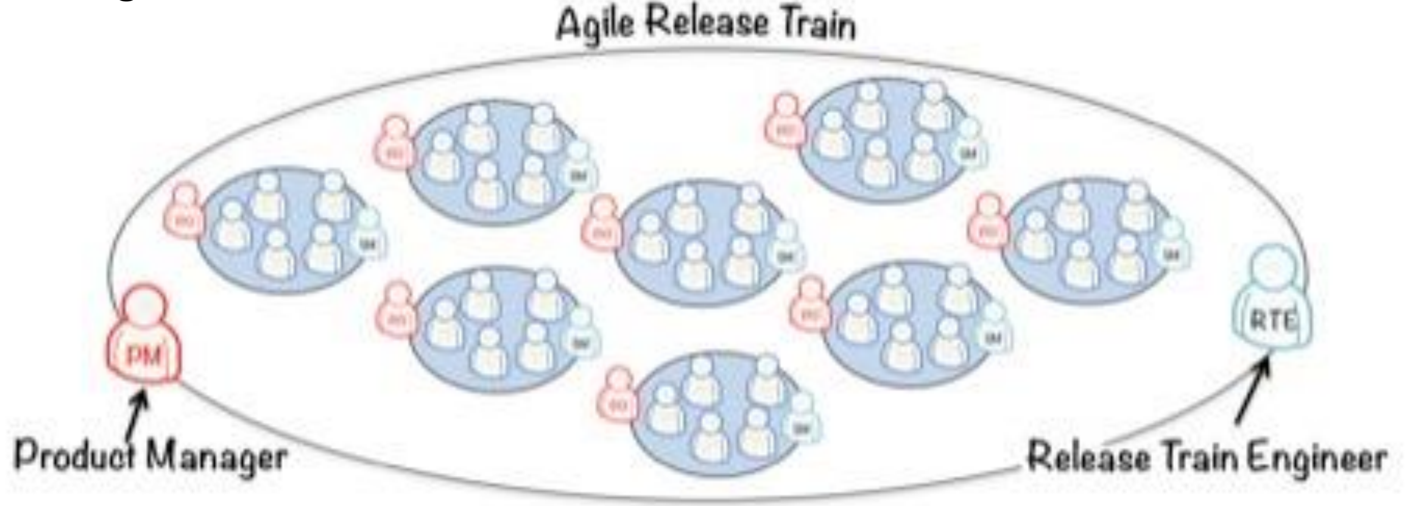


Sprint Review
Team A
Product Owner
Team B
Users and stakeholders
Team Retrospective
Team A
Team B
Overall Retrospective
Manager
Scrum Master
Product Owner
Scrum Master
Team repr.
SM
PO

# SAFe



SAFe's central premise is to divide the work into value streams. A value stream consists of the steps that the company continuously repeats to deliver value to customers and users. Between 5 and 15 teams are typically involved with a value stream, and this grouping of teams is called a release train. A release train can contain up to 150 people. Once that threshold is exceeded it is recommended to have several Agile release trains within each value stream.



# Terima Kasih

## Bibliography

- Garland, Jeff, and Richard Anthony. *Large-scale software architecture: a practical guide using UML*. John Wiley & Sons, 2003.
- Slide Presentation of Jim Fawcett, Dave Penny, "Large-Scale Application Development and Integration-Large Scale Systems – System Decomposition"